10

COMMUNICATIONS

# Optical systems-2

## Fibre optics

We now know that an optically coupled electronic system requires a matching light source and light sensor, and that these must be coupled by a transmission medium in order to operate correctly.

In this chapter, we will look at a single type of optically coupled system, based on a particular transmission medium – optical fibre. (A brief introduction to fibre optics was given in *Communications 1*.)

The principle behind fibre optical communications is that fine strands of transparent material – generally glass – may be used to *guide* light between source and sensor. In this respect, a fibre optic strand acts as a type of **waveguide** (see *Communications 1*).

A single strand of optical fibre consists of an inner **core** of flexible material, covered by a **cladding** layer which, in turn, is covered by a polyurethane jacket. It may only be about 0.1 mm in diameter, or less.

Generally, single strands are not used, instead many strands are combined together into a cable as shown in *figure 1*. Here, about ten fibre strands are wound around a central steel support member.

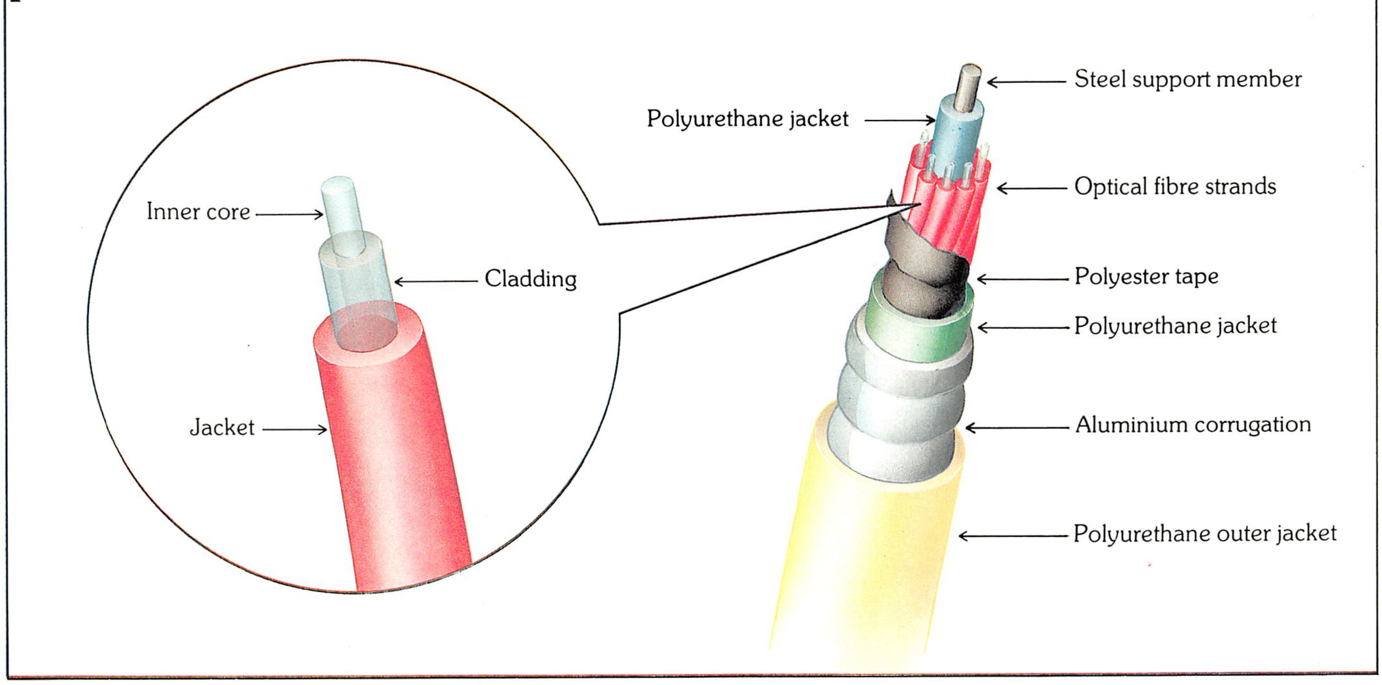
Many different types of these fibre optic cables are available, varying in cost and chosen according to the type of application. There are also a number of different types of optical fibre strands, each with its own characteristics affecting light transmission, but all operating under the same principle.

### Reflection and refraction

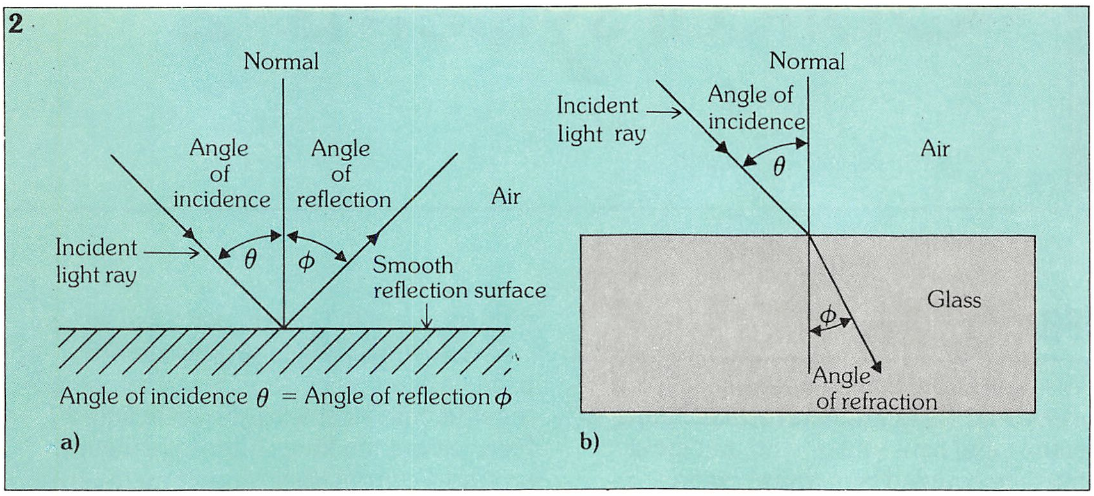
Before the operation of fibre optic strands can be discussed, two important properties of light must be considered: reflection and refraction.

In *figure 2a*, a ray of light is reflected off a smooth surface. The incident light ray strikes the surface at an angle $\theta$ to the

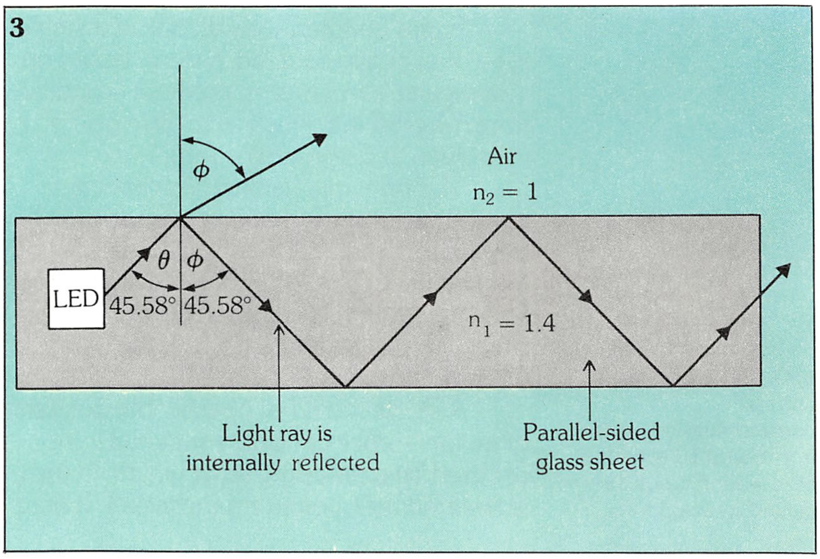
1



**1. Construction of an optical communications cable** from about 10 strands of fibre wound around a central steel support.



**2. (a) For an incident light ray** striking a smooth surface, the angle of incidence = the angle of reflection (within the same material); **(b)** the angle of refraction is different, however, because light travels through different media at different speeds.



**3. Light rays are internally reflected** along a parallel sided glass tube providing they strike the surface at an angle greater than the critical angle.

**normal**. (The normal being an imaginary line drawn perpendicular to the smooth surface.) The angle between the normal and the incident ray is known as the **angle of incidence**.

The reflected light ray leaves the surface at an angle $\phi$ to the normal, known as the **angle of reflection**. Within the same material, say air, *the angle of reflection equals the angle of incidence*.

Consider now the situation shown in *figure 2b*, where an incident ray of light strikes a sheet of glass and is **refracted** through it. Here, the **angle of refraction**, $\phi$, is *not* the same as the angle of incidence, $\theta$ – this is because light travels through different media at different speeds.

The ratio of the speed of light in a vacuum to its speed in a certain material is known as that material's **index of refraction**. A vacuum is chosen as the reference point of 1 because light travels faster through it than through any other medium. The index of refraction, therefore, for any given material will always be greater than 1: for glass it is between about 1.3 and 1.6, depending on the molecular structure; and for air it is so close to 1 that for all but the most accurate calculations, it can be taken as unity.

This relationship between the angles of incidence and refraction can be expressed as **Snell's law of refraction**:

$$n_1 \sin\theta = n_2 \sin\phi$$

where $n_1$ and $n_2$ are the indices of refraction of the two materials through which light is passing.

Snell's law may be used to calculate the angle of refraction, given the angle of incidence and the two indices of refraction. For example, suppose the angle of incidence in *figure 2b* is 30°. The light ray originates in air, where $n_1 \approx 1$, and enters glass, where $n_2 = 1.4$. The angle of refraction may thus be calculated as follows:

$$\begin{aligned} n_1 \sin\theta &= n_2 \sin\phi \\ \sin 30^\circ &= 1.4 \sin\phi \\ \sin\theta &= \frac{\sin 30^\circ}{1.4} \\ &= \arcsin\left(\frac{\sin 30^\circ}{1.4}\right) \\ &= 20.9^\circ \end{aligned}$$

Therefore, the light ray which enters the air/glass interface at 30° from the normal, leaves it at 20.9° off the normal.

### Critical angle

If we now consider an LED light source which is placed inside a parallel-sided glass sheet, as shown in *figure 3*, we see that light leaving the LED strikes the inside surface of the glass at an angle $\theta$, and exits at an angle $\phi$. Snell's law states that:

$$1.4 \sin\theta = 1 \sin\phi$$

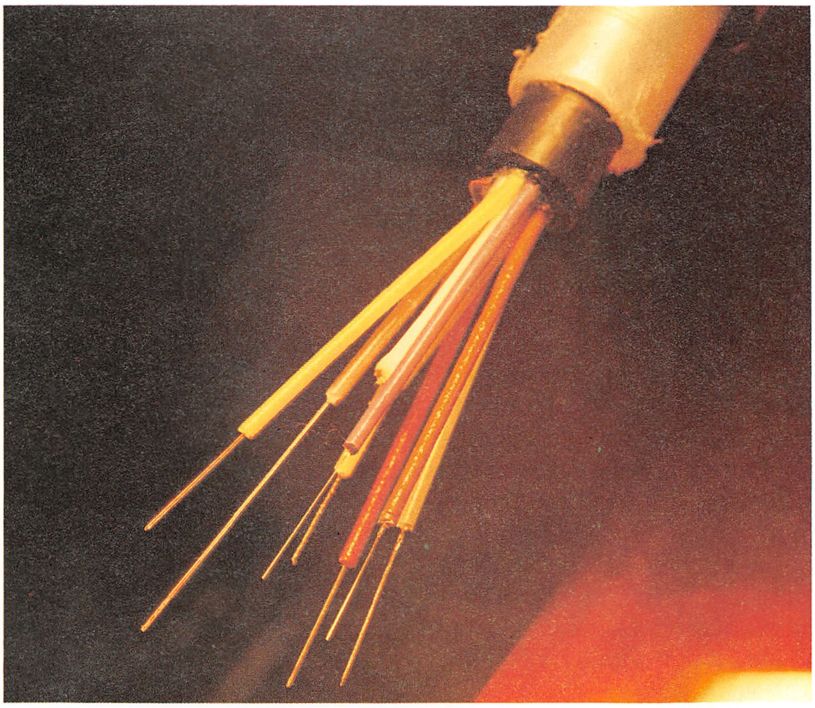
where the refractive index for glass is 1.4.

We can see from *figure 3* that if $\phi \geqslant 90°$, light remains *inside* the glass sheet. A calculation using Snell's law shows that when $\phi$ is 90°:

$$\begin{aligned} 1.4 \sin\theta &= 1 \sin 90° \\ 1.4 \sin\theta &= 1 \times 1 \\ \sin\theta &= \frac{1}{1.4} \\ \theta &= \arcsin(0.714) \\ \theta &= 45.58° \end{aligned}$$

This is known as the **critical angle**. All light striking the surface at an angle greater than the critical angle (45.58° in this example) is *internally reflected* at an angle equal to the angle of the incidence, according to the rules of reflection. Since the surfaces of the sheet are parallel, the beam is reflected at the same angle from the opposite surface, and so on. As you can see, the parallel surfaces of the glass form a *light guide*.

**Below:** close up of 8-fibre communications cable.



Paul Brierley

## Optical fibre strands

Optical fibre strands can be considered as special cases of light guides, and can be produced with extremely good transmission efficiencies. The strand shown in *figure 4* comprises a cylindrical core, with an index of refraction of 1.5, surrounded by cladding, with an index of refraction of 1.4. The critical angle can be calculated using Snell's law to approximately 69°:

$$\begin{aligned} 1.5 \sin\theta &= 1.4 \sin 90° \\ \sin\theta &= \frac{1.4 \times 1}{1.5} \\ \sin\theta &= \arcsin(0.933) \\ \theta &= 68.96° \end{aligned}$$

Light rays entering the fibre at angles greater than 69° are internally reflected, remaining inside the strand until they exit at the other end. Bends in the fibre which are not abrupt do not significantly alter the efficiency of light movement, and so fibres can be used to 'pipe' light over long distances with little loss. By controlling the medium, therefore, light signals can be delivered from relatively low intensity sources over great distances to a sensor.

It may also be observed that light entering the fibre from the side, passes straight through without being internally reflected, thus reducing interference from unwanted signals.

### Types of optical fibres

There are two main categories of optical fibre: **multimode** and **monomode**.

A length of multimode fibre can be seen in *figure 5a*. In this type of fibre, light rays can follow many different paths or **modes** as they travel from source to sensor – hence the name. In fact, many hundreds of modes are possible.

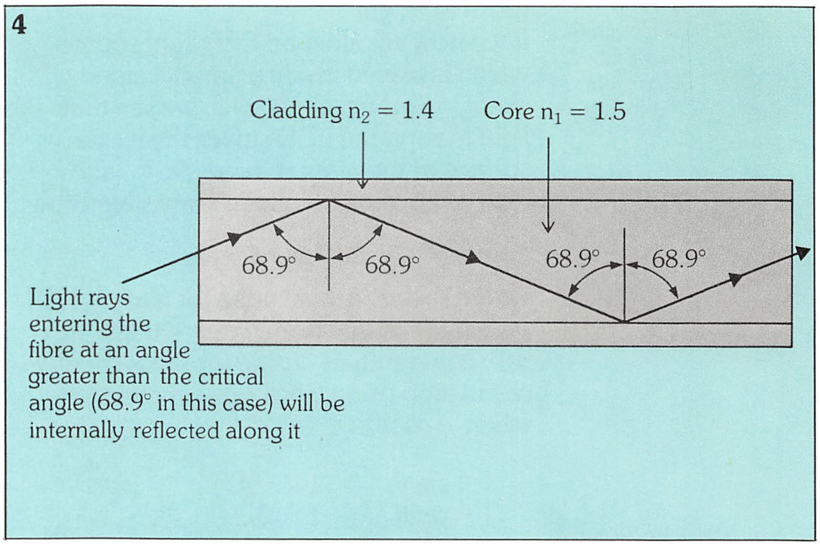
This, however, causes problems known as **modal dispersion**. Some modes will obviously be longer than others, and so different light rays travelling along the fibre will arrive at the sensor at different times, meaning that the information that the light rays represent will be distorted – the amount of distortion increasing with fibre length. This type of multimode fibre can therefore only be used in lengths up to about 10 km.

This problem can be overcome by

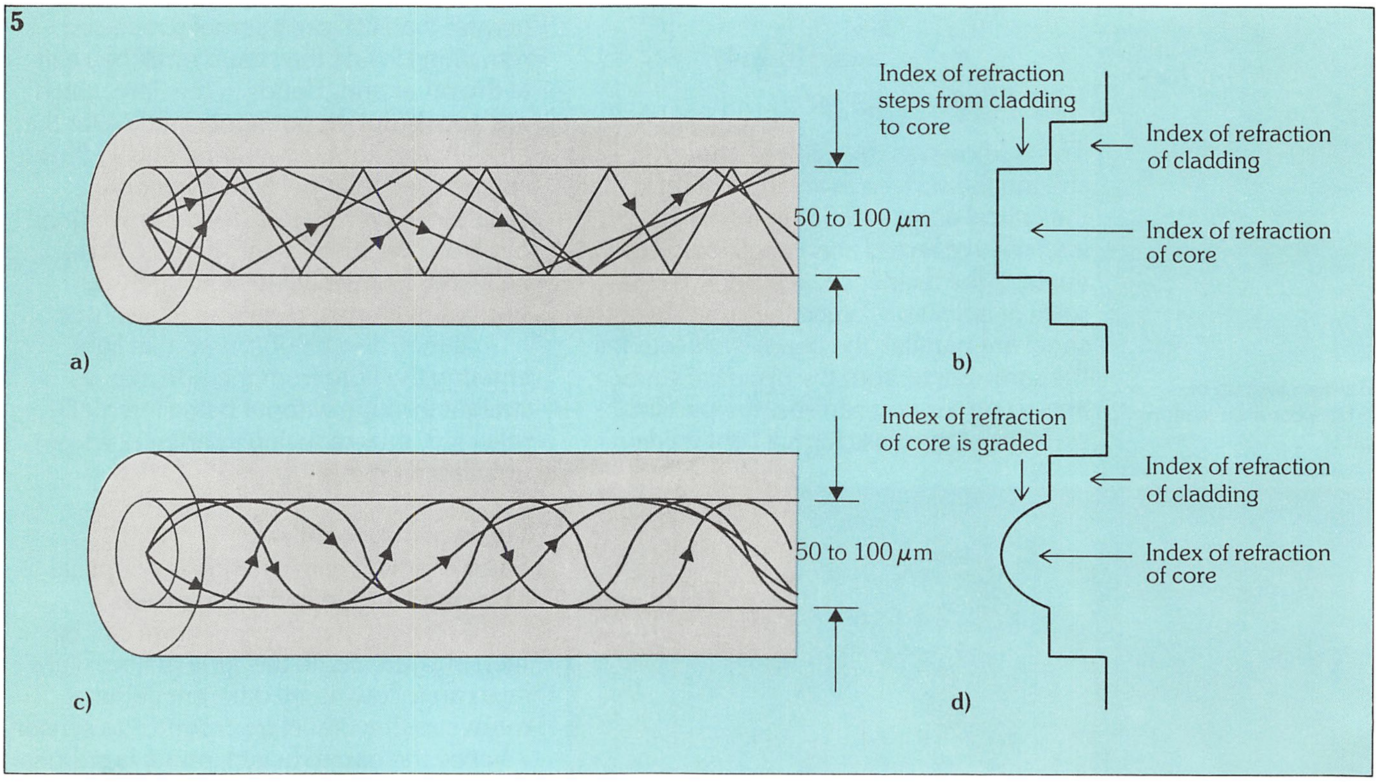
using different types of multimode fibre. The **refractive index profile** of the fibre in *figure 5a* is shown in *figure 5b*. Here, the difference in the indices of refraction between core and cladding is shown graphically. As this multimode fibre has a stepped increase of index of refraction between cladding and core, it is known as **stepped-index multimode fibre**.

A different type of multimode fibre, with a different refractive index profile, is shown in *figures 5c* and *5d*. You'll notice here that the refractive index profile has a *varying* core index of refraction from the cladding to the centre of the fibre and is therefore known as **graded-index multimode fibre**.

4


Cladding $n_2 = 1.4$
Core $n_1 = 1.5$
68.9°
68.9°
68.9°
68.9°
Light rays entering the fibre at an angle greater than the critical angle (68.9° in this case) will be internally reflected along it

5


Index of refraction steps from cladding to core
Index of refraction of cladding
50 to 100 $\mu$m
Index of refraction of core
a)
b)
Index of refraction of core is graded
Index of refraction of cladding
50 to 100 $\mu$m
Index of refraction of core
c)
d)

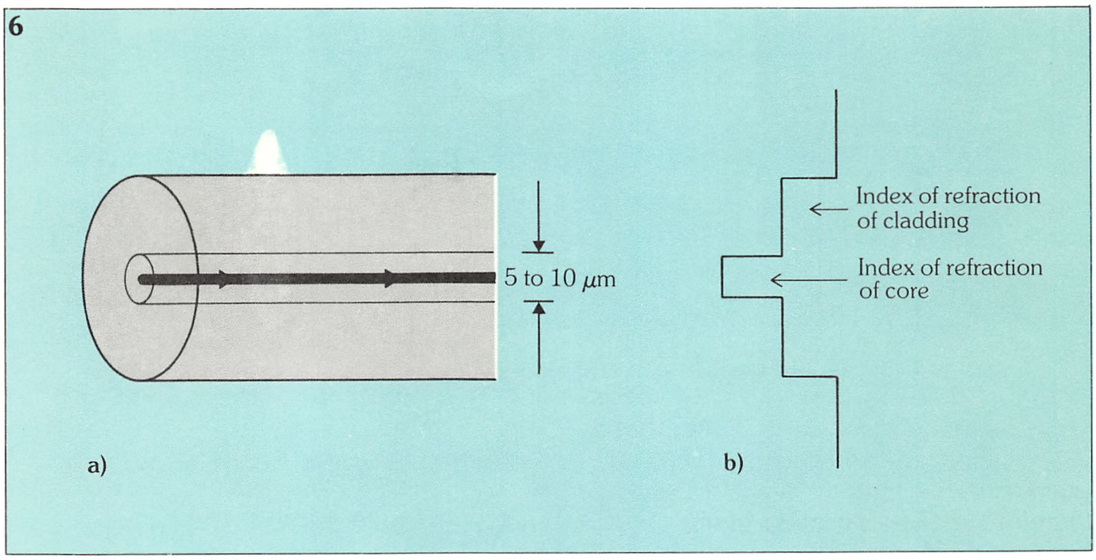
As the core's index of refraction varies, so the speed of the light travelling through it also varies – light in the centre of the core travelling more slowly than light closer to the cladding. The overall effect of this varying index is that light travelling in the various modes take about the same time to be transmitted along the fibre. Distortion is therefore reduced and information may be transmitted over longer lengths of fibre – up to about 50 km or so, compared with the 10 km for stepped-index fibre.

By reducing the diameter of the fibre core, so that it is little greater than the wavelength of the light ray to be transmitted, optical fibres may be made which allow only one mode of light. Such a **monomode**, or **single mode**, optical fibre is shown in *figure 6a*, with its refractive index profile in *figure 6b*. Distortion due to different modal lengths is eliminated, and so fibre lengths may be 100 km, and over, with ease.

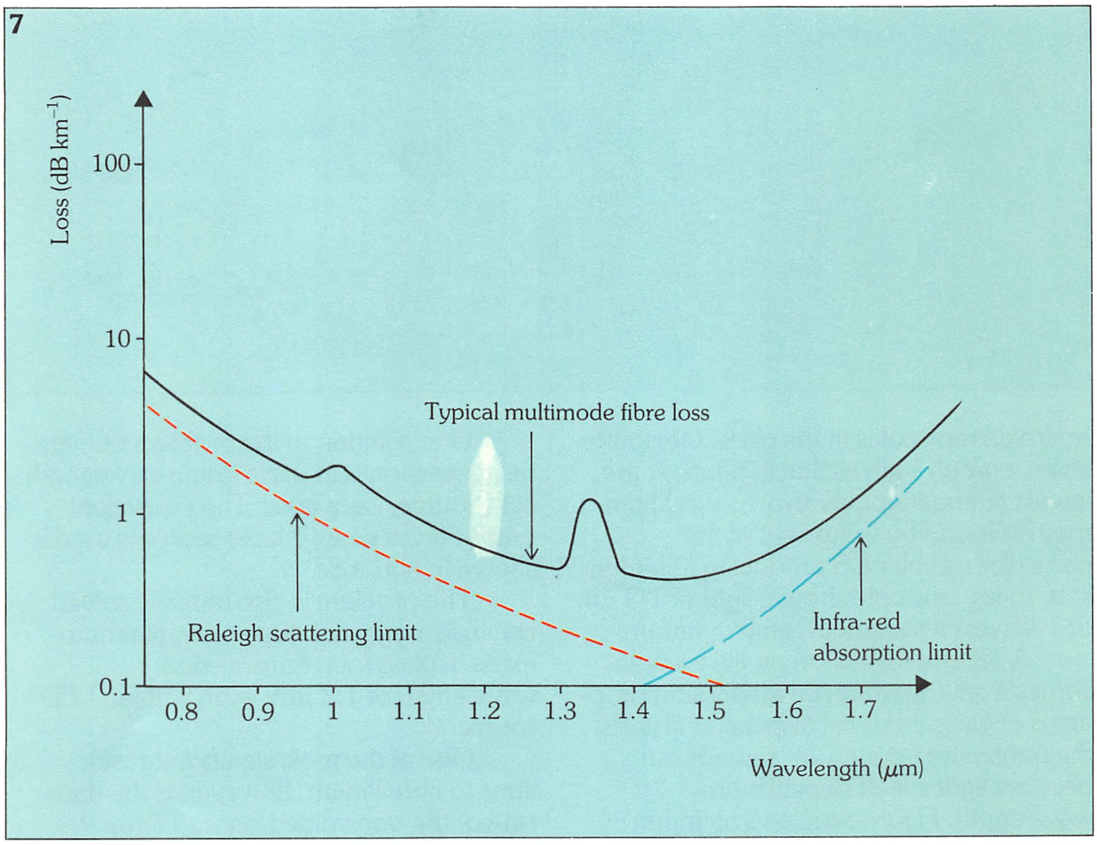
**4. Optical fibre strands act as light guides,** piping light over considerable distances with little loss.

**5. Two different types of multimode fibre: (a)** stepped-index fibre; **(b)** its refractive index profile; **(c)** graded-index fibre; **(d)** its profile.

**6. (a) Monomode optical fibre** has such a narrow diameter that only one mode of light is possible; **(b)** its refractive index profile.



**7. Graph of loss *vs* wavelength** for a typical graded index multimode fibre.

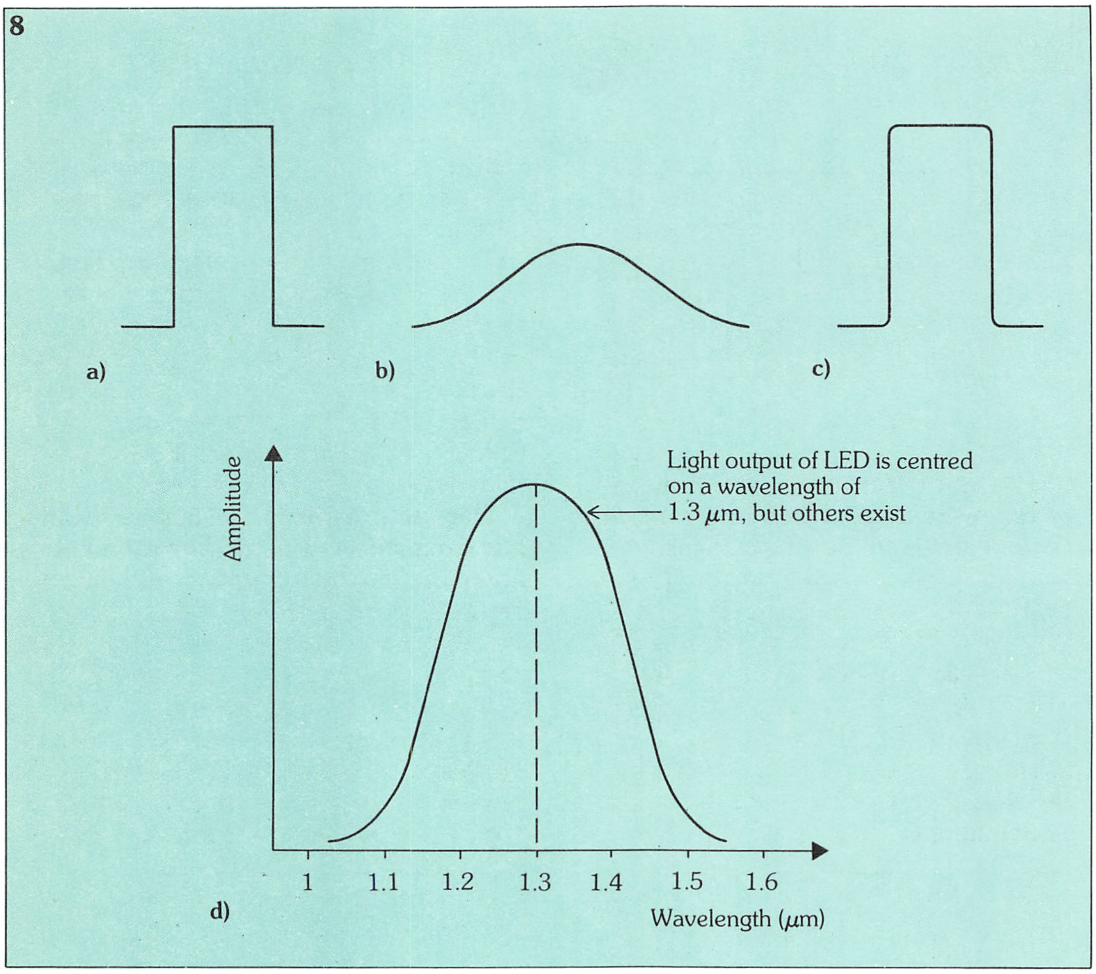


## Other restrictions on fibre length

The distortion due to a varying modal length is not the only restriction on the length of optical fibre that can be used in a communications system. Attenuation of the transmitted signal, for example, is an important criterion.

*Figure 7* shows a graph (in solid line) of loss against light wavelength for a typical graded-index multimode fibre. The red broken line indicates the minimum loss possibly attainable, due to a phenomenon known as **Rayleigh scattering**, caused by the intrinsic impurities of the glass. The blue broken line indicates the minimum loss due to infra-red absorption of light by conversion to heat.

Modern optical fibre loss can be seen to approach these limits, apart from a few minor irregularities due to the presence of


8
a)
b)
c)
Amplitude
Light output of LED is centred on a wavelength of 1.3 $\mu$m, but others exist
1
1.1
1.2
1.3
1.4
1.5
1.6
Wavelength ($\mu$m)
d)

**8. The material dispersion properties of glass** tend to distort the transmitted signal shown in **(a)** to that shown in **(b)**; **(c)** the equivalent output pulse if a laser source was used; **(d)** graph of amplitude *vs* wavelength illustrating this phenomenon.

hydroxyl molecules in the glass. Obviously, given a graph such as that in *figure* 7, the lowest attenuation due to an optical fibre may be found by using light of the particular wavelength producing minimum loss. In the example shown, light of 1.3 $\mu$m or 1.45 $\mu$m wavelength is the optimum.

A second restriction on fibre length comes from the **material dispersion** properties of glass. Material dispersion distorts the transmitted signal – it 'spreads out' because light speed depends on wavelength. *Figure 8a* shows an input pulse from an LED source to an optical fibre; the output pulse from the same fibre after, say, 50 km is shown in *figure 8b*.

The reason behind this phenomenon is shown in *figure 8d*. The light output from an LED, although centred on a particular wavelength, also comprises other wavelengths. Each wavelength of light is transmitted at a slightly different speed along the fibre, so the longer the fibre, the greater the signal distortion.

One solution to the problem of material dispersion is to use a single wavelength light source, i.e. a laser. The equivalent output pulse when a laser source is used is shown in *figure 8c*.

The problem is also partially solved because a material dispersion minimum exists in glass for a transmission wavelength of 1.3 $\mu$m, even with an LED source.

One of the most significant restrictions to fibre length, however, is the **data rate** of the transmitted signal. *Figure 9* shows a graph of data rate against fibre length for different optical fibres. From this can be seen how light wavelength, index of refraction, light source, fibre mode and data rate affect possible fibre length.

It is interesting to note that the potential of monomode fibre has not yet been fully exploited and higher data rates over longer lengths will exist in the near future. Laboratory tests have already shown data rates of over 5 Gbits$^{-1}$.

## Fibre optic sources and sensors

Generally, infra-red LEDs are used as sources for multimode fibres in short distance communications links and laser sources are used in higher data rate, longer length multimode links. Lasers alone are used as sources in monomode fibre links, where narrow spectral width is essential.

Although a source wavelength of 1.3 $\mu$m is, as we have seen, the optimum in terms of attenuation and dispersion, LEDs and lasers are available in the **short wavelength** range of 0.85 $\mu$m to 0.9 $\mu$m. These sources are generally of gallium arsenide (GaAs) semiconductor. Research is also taking place into long wavelength (i.e. 1.5 $\mu$m) sources, using GaInAsP.

Avalanche photodiodes, or p-i-n diodes are used as sensors. These short wavelength sensors are generally of silicon construction. Sensors in the longer wavelength ranges are made from germanium, GaInAsP or GaAlAsP.

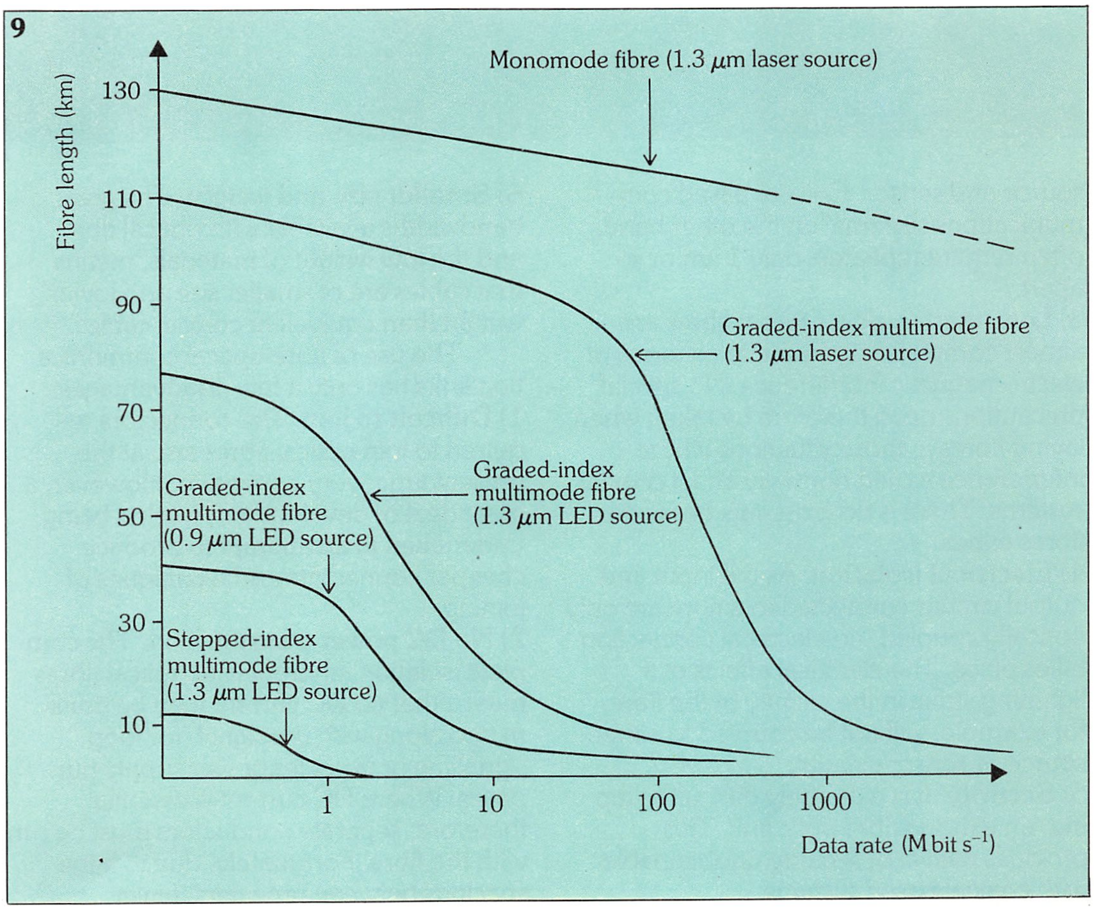
## Advantages and disadvantages

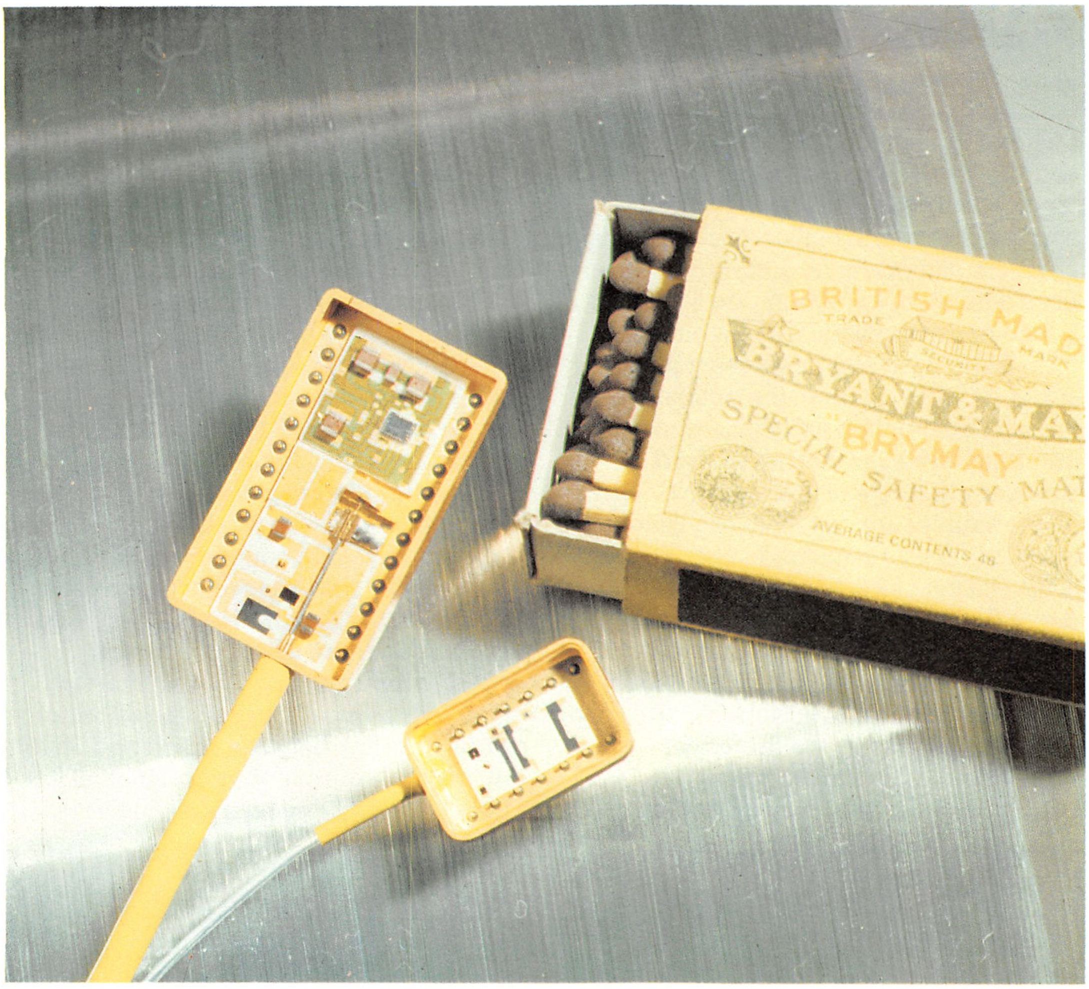
A fibre optic communications system has many advantages over traditional, coaxial wire based systems.

1) **Data capability**. The extremely large bandwidth available with optical fibres means that many different signals can be multiplexed onto a single fibre (many more than can be multiplexed onto a coaxial cable). Voice, data and television signals may all be combined together and transmitted over a single optical link.

2) **Repeater distance**. Because of the low loss experienced with modern monomode fibres, and the high coupling efficiency of laser sources, the distance between repeaters is far lower than that in coaxial based systems. The fibre length of a typical optical communications system may be many tens of km – the recently introduced optical fibre link between Luton and Milton Keynes, for example, carries signals at 140 Mbit s$^{-1}$ without any repeaters between

**9. Data rate *vs* fibre length** for various optical fibres.





Left: connectors used in optical communications interfacing optical fibre with electrical systems.

The Research House/British Telecom

source and sensor. Coaxial based communications systems, on the other hand, often require repeaters only 1 km or so apart.

3) **Low interference**. Optical fibres are almost completely immune to all forms of electromagnetic interference. No special precautions need therefore be taken when laying fibres in those situations where interference would normally affect coaxial systems. There is no cross-talk between fibres either.

4) **Electrical isolation**. As the input and output circuits connected to a fibre are only optically coupled, no electrical connection takes place. The electrical effects of a lightning strike in the vicinity of the fibre, for example, will not be coupled to either source or sensor circuitry.

5) **Security**. It is extremely difficult to tap into an existing fibre optic link. This provides a level of security unobtainable with coaxial based systems.

6) **Smaller size and weight**. The great bandwidths available with optical fibre, and the low weight of materials, means that cables are of smaller size and lower weight than equivalent coaxial cables.

The use of fibre optical communication links has only a few disadvantages:

1) **Difficult to join**. The connectors required to join optical fibres are, at the present time, very expensive. However, a great deal of development work is being undertaken in an attempt to produce cheaper connectors and techniques of joining.

2) **No DC power transmission**. The complete isolation inherent with optical fibres means that no DC current may be transmitted along with the signal, for loop signalling or repeater power supply purposes. Where DC current is essential, therefore, separate conductors must be run with the fibre. Fortunately, due to large spacings between repeaters, fewer

repeaters are required anyway.

### Into the future

Fibre optical communications is still a developing area and the great data rate capabilities of current monomode fibres will undoubtedly be surpassed in the future.

One of the most promising new developments is **wavelength multiplexing** of signals onto a single fibre. That is, the use of more than one source wavelength to multiplex together signals which would normally be transmitted over separate fibres. For example, using source wavelengths of, say, 0.85 μm, 1.3 μm and 1.5 μm, three signals may be transmitted over a single fibre, thus potentially tripling the fibre's data capability.

Fibres are generally constructed from very pure glass, thus producing very low attenuation levels specifically meant for long distance communications links. (The complete Luton to Milton Keynes fibre link mentioned earlier, for example, produces light attentuation equivalent to only a few centimetres of ordinary windowpane glass.) Where higher levels of attenuation are acceptable however, for instance on local lines between a telephone terminal and the local exchange, *plastic fibres* may be suitable. Such **polymer fibres** will be many times cheaper than the present glass fibre.

It is easy to see, therefore, that optical fibre will play a major role in future communications systems.

## Glossary

| | |
|---|---|
| **cladding** | outer layer of transparent material in an optical fibre. The index of refraction of the cladding is lower than that of the inside core |
| **core** | inside of an optical fibre |
| **graded-index multimode fibre** | optical fibre whose core has a graded index of refraction |
| **material dispersion** | dispersion of light within an optical fibre due to the varying speed of light with different wavelengths |
| **modal dispersion** | dispersion of light within an optical fibre due to the different modes that light rays may take along the fibre |
| **mode** | path of a beam of light along an optical fibre |
| **monomode or single mode, optical fibre** | optical fibre whose core is of such small dimension that only one path or mode of light is possible |
| **multimode optical fibre** | optical fibre in which many modes of light are possible |
| **Rayleigh scattering** | scattering of light rays within glass due to inherent impurities. Rayleigh scattering sets a lower limit on the attenuation of light along optical fibre |
| **refractive index profile** | method of graphically illustrating the difference in indices of refraction between core and cladding |
| **step-index multimode fibre** | optical fibre whose indices of refraction steps up from the cladding to the core |
| **wavelength multiplexing** | possible future use of optical fibres where more than one wavelength of light is used to multiplex signals onto an optical fibre |

5 MICROPROCESSORS

# The instruction set

## Status bits

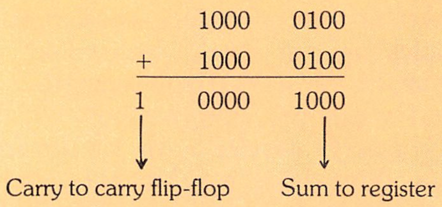
In *Microprocessors 4* we began a discussion of the microprocessor's instruction set and noted that there are two basic types of instruction – arithmetic and data movement. Before going on to look at the other types of instruction – shift, logical, comparison and branch – we'll return to artithmetic instructions to find out about the indicators known as **status bits** or **flags**.

### Status bits – zero and sign

In order for the microprocessor to keep track of the result of an arithmetic operation, such as addition or subtraction, some condition of that result must be stored. As a zero result may be important, a bit indicating this is stored in a register – the **zero flip-flop**. Referring to the contents of the zero flip-flop (with later instructions) therefore checks whether or not the result of a given operation was zero. Any subsequent actions are then determined by this fact.

In a similar way, storing the sign of a number in a **sign flip-flop**, enables checking to see if the result of an operation was positive or negative.

1



**1. A carry-out** resulting from a bit sum is stored as a carry status bit.

Table 1
**Positive and negative numbers with 8-bit code**

| 8-bit binary code | Decimal equivalent of magnitude code | Decimal equivalent of two's complement code |
|---|---|---|
| 0000 0000 | 0 | +0 |
| 0000 0001 | 1 | +1 |
| 0000 0010 | 2 | +2 |
| . | . | . |
| . | . | . |
| . | . | . |
| 0111 1111 | 127 | +127 |
| 1000 0000 | 128 | −128 |
| 1000 0001 | 129 | −127 |
| 1000 0010 | 130 | −126 |
| . | . | . |
| . | . | . |
| . | . | . |
| 1111 0001 | 241 | −15 |
| 1111 0010 | 242 | −14 |
| . | . | . |
| . | . | . |
| . | . | . |
| 1111 1111 | 255 | −1 |

Zero and sign flip-flops are two examples of status bits or flags which, along with other condition flip-flops, reside in the **status register** of the microprocessor.

### Status bits – carry and borrow

When two N-bit numbers are added and a carry-out results from the bit sum (*figure 1*), this fact is stored as the **carry status bit** in the status register. Similarly, if two numbers are subtracted and a borrow is needed, this fact is also stored in the carry status bit.

The carry or borrow status is then passed on to higher order digit positions in addition or subtraction. Many microprocessors provide add-with-a-carry and subtract-with-a-borrow instructions to ease handling of carries and borrows in problems using multiple 8-bit groups (multi-byte problems).

### Status bit – overflow

Some microprocessors provide an **overflow status bit** as a given N-bit binary number can only express numbers within a certain range. For example, *table 1* shows that with two's complement notation, an 8-bit code can only represent numbers from –128 to +127; the sum of say, 100 + 100 = 200 cannot be properly represented by this code (and neither could –100 – 100 = –200).

If the result is too large for the 8-bit code then it can be seen to 'overflow'. An **overflow flip-flop** is set up in the status register to record a 1 if this occurs, so the microprocessor can take corrective action.

## Shift instructions

We know that it is possible to multiply or divide a binary number by two, by shifting it one bit position to the left or right respectively. *Figure 2* illustrates how the end bits are inserted during the shifting – this is essential when dealing with two's complement numbers.

To divide a positive binary number by two, a 0 is shifted into the vacated most significant bit position. To divide a negative two's complement number by two, a 1 is shifted into the vacated leftmost position.

### Shift right

Depending upon the microprocessor, there are usually two or three different shift operations from which to choose, as shown in *figure 3*.

A **logical shift right** moves the least significant bit into the carry flip-flop and shifts in a 0 from the left. This operation is used when dividing an unsigned binary number by two, or when examining the least significant bit of a number.

The **arithmetic shift right**, on the other hand, also moves the least significant bit into the carry flip-flop but shifts the sign bit ($d_7$) into the next bit to the right ($d_8$). It also leaves a copy of the sign bit in its original bit position ($d_7$). This shift operation divides a signed (positive or negative) two's complement number by two.

The **circulate right** operation sends the carry flip-flop value into the left side of the number, and shifts the least significant bit into the carry flip-flop. This operation is

**2. Effect of a right shift** on binary numbers.

**2**

| | | | |
|---|---|---|---|
| | Shifting the binary equivalent of +28 to right one bit position: | | |
| Before shift: | 0001 1100 | | (28) |
| After shift: | 0000 1110 | | (14) |
| | | Carry=0 | |
| | Shifting the binary equivalent of −28 to right one bit position: | | |
| Before shift: | 1110 0100 | | (−28) |
| After shift: | 1111 0010 | | (−14) |

useful when a shift is performed on a multiple byte group of binary digits.

### Shift left

Most microprocessors carry the two shift left operations shown in *figure* 4. The **logical shift left** instruction moves the leftmost bit into the carry flip-flop and shifts a 0 into the empty least significant bit position. This instruction can be used to multiply a number by two, or examine the value of the most significant bit ($d_7$).

A **circulate left** shifts the value in the carry flip-flop into the empty least significant bit position. The bit that is shifted out of the most significant bit position is stored in the carry flip-flop. This operation is useful for performing a shift left on multiple groups of words.

## Logical instructions

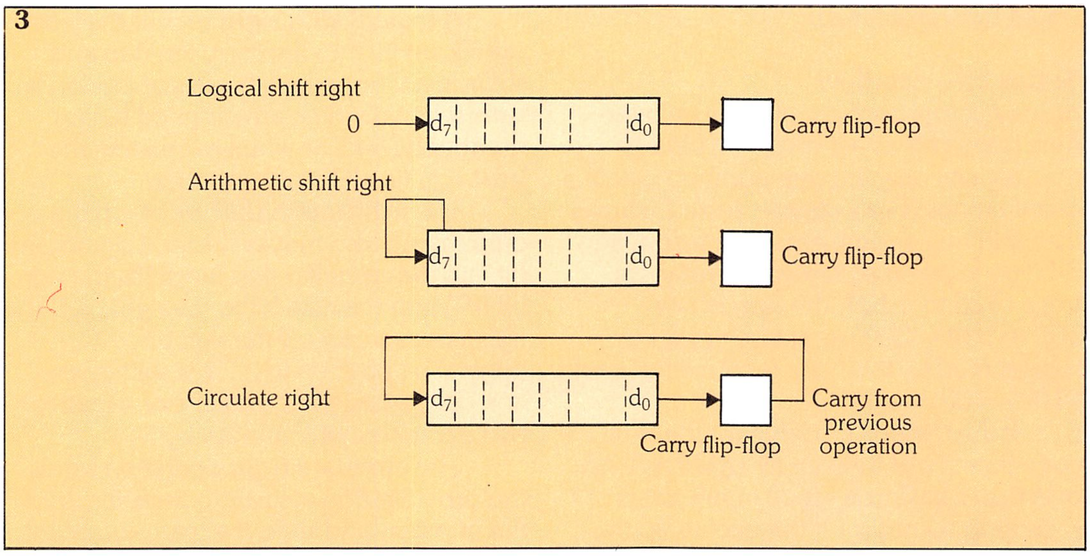
Microprocessors usually offer the OR, AND and NOT logical instructions on a bit by bit basis. What this means (as shown in *figure* 5) is that each logical operation is performed as if the input was fed into a two-input gate, one bit position at a time.

The logical operation exclusive OR is also usually provided, and is usually represented by the mnemonic XOR.

### The NOT function

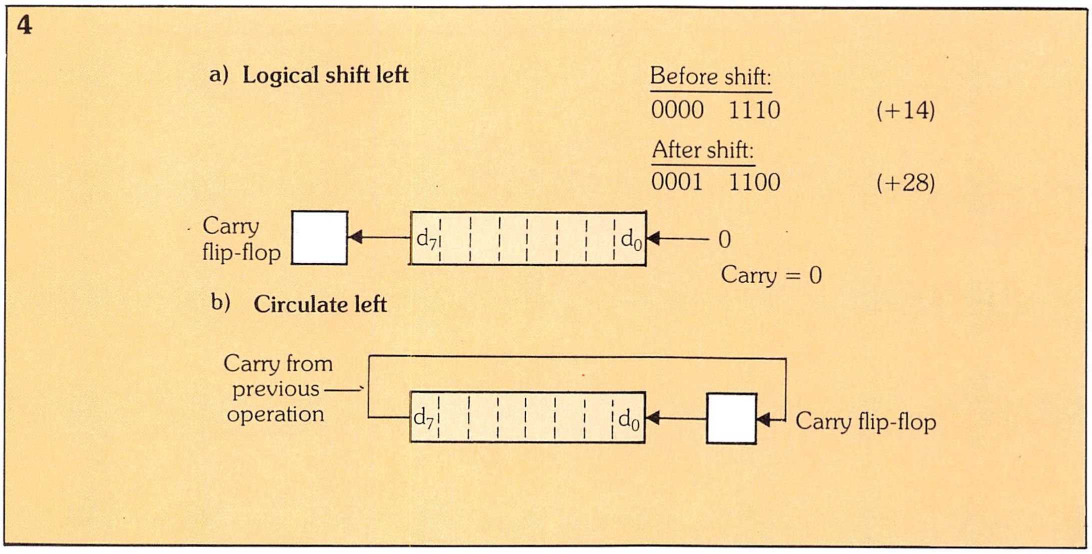
The NOT operation is used to create the two's complement of a binary number. This can be found by inverting all the bits and adding one to the answer (as shown in *figure* 6).

3



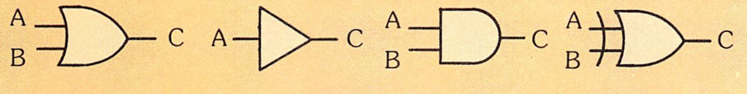
**3. The three microprocessor shift right** operations.

4



**4. The two shift left** operations.

5



| | | OR | NOT | AND | XOR |
|---|---|---|---|---|---|
| Inputs: | A | 1010 | 1010 | 1010 | 1010 |
| | B | 1100 | | 1100 | 1100 |
| Result: | C | 1100 | 0101 | 1000 | 0110 |

**5. The four microprocessor logical** operations.

6

| | | | | |
|---|---|---|---|---|
| Binary number A: | 0010 | 1001 | 0101 | 1010 |
| Complement (NOT) of A: | 1101 | 0110 | 1010 | 0101 |
| Add 1: | 0000 | 0000 | 0000 | 0001 |
| Result is two's complement of A: | 1101 | 0110 | 1010 | 0110 |

**6. The use of the NOT operation** to form the two's complement of a binary number.

7

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Light number: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
| Initial output: | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1-On 0-Off |
| Output after ORing with 0000 1001: | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | |

Lights 1 through 4 are left on; lights 6 and 7 are left off; and lights 5 and 8 are turned on.

**7. The use of the OR function.**

8

| | | | |
|---|---|---|---|
| BCD code for 93 | 1001 0011 | 1010 1101 | Input code |
| Masking code | 0000 1111 | 0000 0001 | AND operation |
| Result of AND operation | 0000 0011 | 0000 0001 | Result $d_0=1$ |
| Decimal equivalent | 0 3 | | |
| **a) Masking** | | **b) Examining $d_0$** | |

**8. The use of the AND function in: (a)** masking; and **(b)** examining bits.

### The OR function

The OR operation sets certain bits to 1, without affecting other bits in a binary code. An example is shown in *figure 7* where a system has eight lights that can be turned on (output a 1) or off (output a 0). These lights are controlled by an 8-bit binary code. At first, lights 1 to 4 are turned on. In the next step, these are left on, but lights 5 and 8 are also switched on.

An instruction to OR the binary number 0000 1001 with the current on-off code of 1111 0000 results in the output code 1111 1001. This code controls the lights, so that the first four lights are left on, lights 5 and 8 are turned on, and lights 6 and 7 are left off.

### The AND function

The AND operation is used to mask out certain bits of a binary number. As an example, take the BCD code 1001 0011, which represents the decimal number 93 (*figure 8a*). Suppose that the 3 is the only digit of interest – using an instruction to AND the binary code 0000 1111 with 1001 0011, the four most significant bits will be turned to zeros and masked off.

This effectively isolates 03 from 93.

The AND operation can also be used to examine a single bit of a binary number. For example, the least significant bit of the number shown in *figure 8b* can be examined by ANDing the binary number with 0000 0001 – this zeros the first seven bits and leaves the last bit ($d_0$) unchanged. If $d_0$ is 0, then the zero flip-flop is set to 1; if $d_0$ is 1, then the zero flip-flop will be cleared to 0. Instructions that refer to the contents of the zero flip-flop later on in the program can then react accordingly.

### The XOR, used for comparison

Returning to *figure 5*, you can see that each XOR result would be 0, if all the bits in both binary numbers were the same. If any bits are different, a 1 results in the relevant positions. All resulting bits are latched into a register at the same time and then shifted out. If a 1 appears at the output during the shift, then the two numbers were different, and action based on this fact can then be initiated.

The logical operations that we have seen have more complicated applications which are variations on these examples.

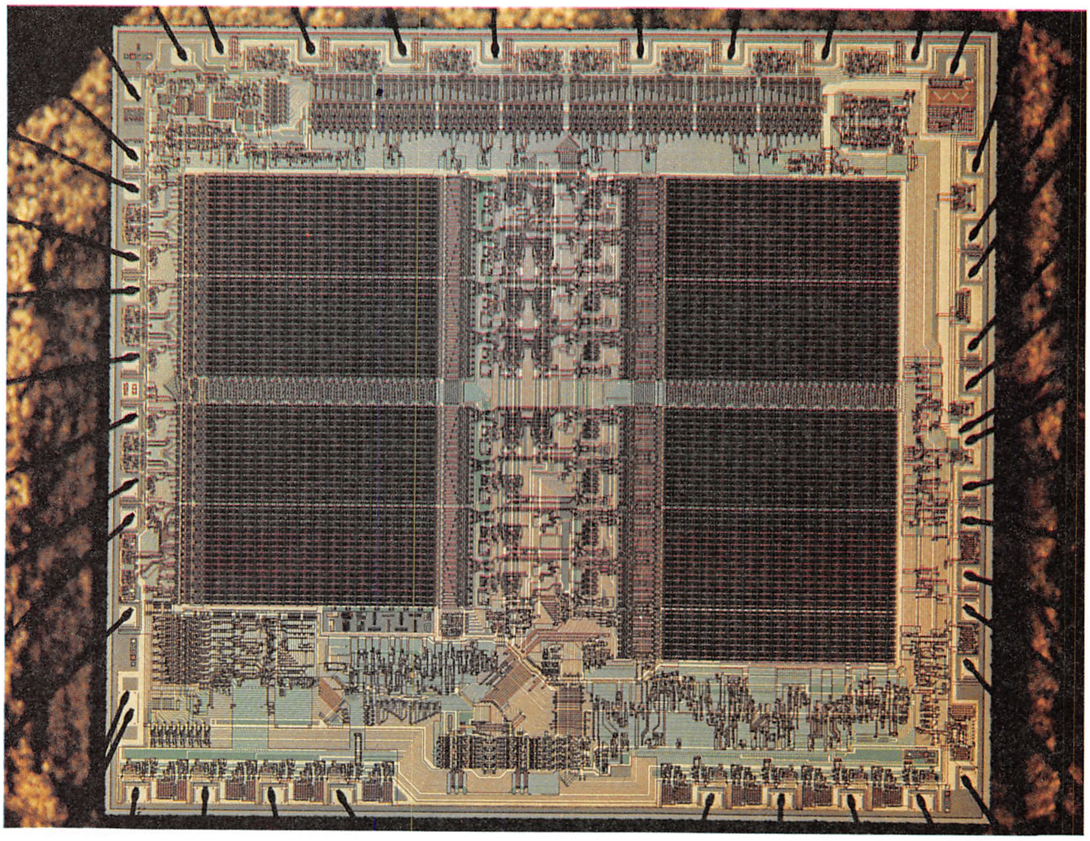
## Comparison instructions

Microprocessor instruction sets usually contain a number of separate comparison instructions because many tests and comparisons must be made on binary numbers, without actually changing them during the process.

Comparison instructions generally cause the second number involved in the comparison to be subtracted from the first. The subtraction merely changes the status of the condition code flip-flops, but does not change either of the two numbers involved in the comparison operation.

The result of the comparison can be used later, by checking the condition code flip-flops. The most common checks are to the zero flip-flop, to see if the two numbers or codes compared were the same, and to the sign flip-flop, to see if the first number was greater than or equal to the second number.

Using these comparisons with the related conditional branch instructions, step by step sequences of instructions can be written. These are then able to make complex decisions.



**Left:** photomicrograph of CMOS memory chip. (Photo: SGS).

## Branch instructions

From *Microprocessors 1* we know that the microprocessor executes the instructions addressed by the program counter and, unless told otherwise, it takes instructions from memory in order – one after another. This occurs because, normally, the program counter is automatically incremented to the next instruction in sequence.

Sometimes, however, the operation has to be transferred to a sequence of instructions held somewhere else in memory. For example, when an interrupt occurs and needs servicing. Instead of the program counter being incremented automatically to point to the next instruction in sequence, it is loaded with a new address which points to an instruction held elsewhere in memory. The instruction that performs this is known as a **branch** or **jump** because the program jumps to a new location in memory for the next instruction. Further instructions are taken in sequence from that address until a new branch or jump is encountered. This usually returns control to the original part of the program.

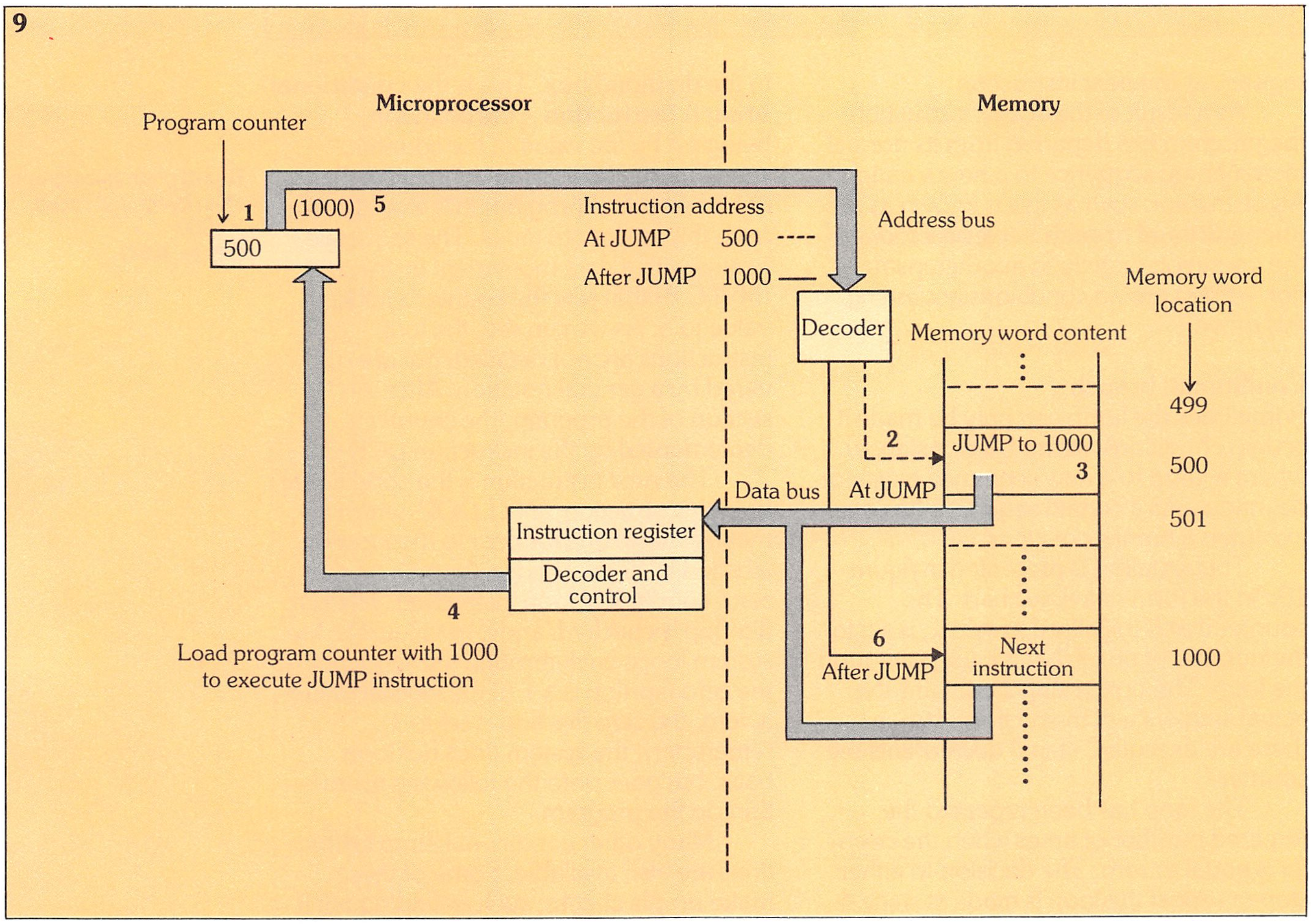
**Unconditional branch**

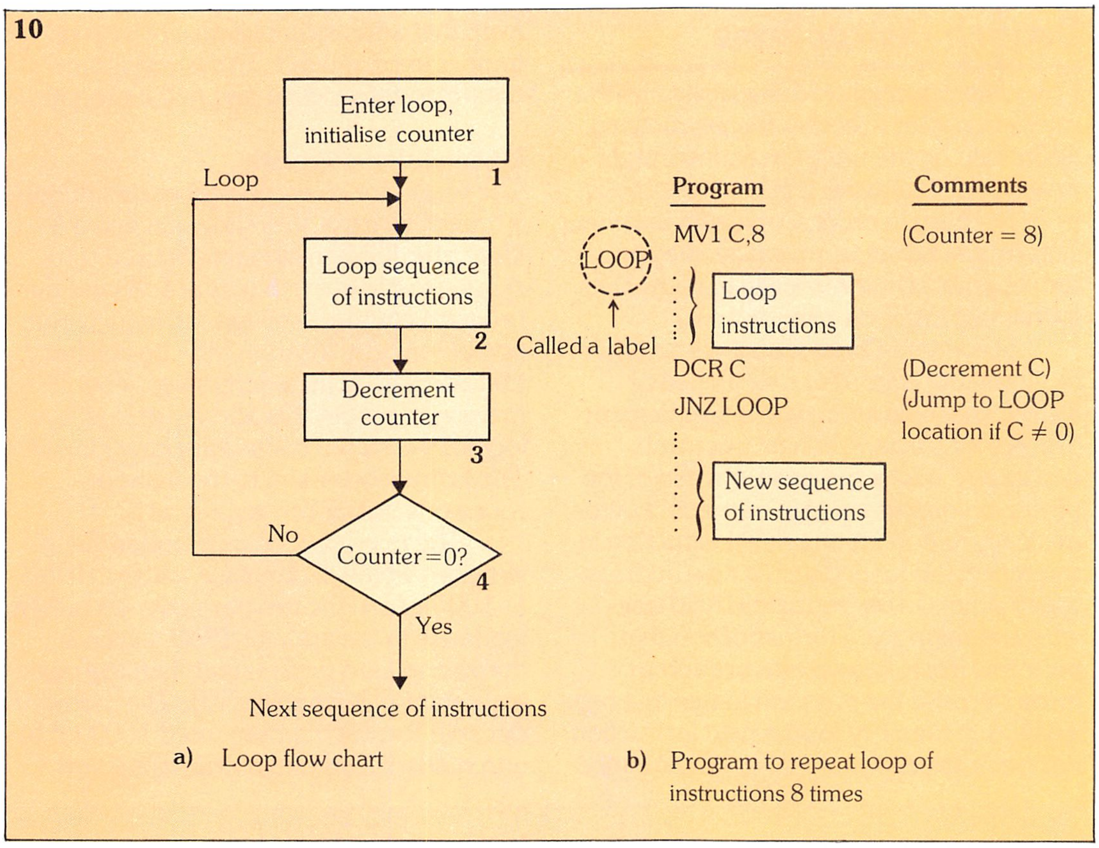
The sequence of events that occurs during a branch instruction is shown in *figure 9*. Here, the JUMP mnemonic is used. The sequence of events begins at **1**, where the program counter requests the next instruction at memory location 500, by placing that address on the address bus. The memory decodes this address and at **2**, locates the stored contents at location 500. The instruction is sent to the microprocessor's instruction register at **3**.

The microprocessor decodes the instruction which, in this case, means 'JUMP to 1000'. At **4**, the program counter is then loaded with the address 1000, which is then jumped to (at **5**) rather than the next incremented instruction 501. The memory decoder locates memory word 1000 (at **6**) and reads its contents to the instruction

**9. Sequence of events during a branch** instruction.

9





**10. Use of conditional branch** instructions.

**11. Subroutine jump** operation.

register, as the next instruction.

As a result of the JUMP instruction, the program has departed from its normal incremented sequence to a new location. An instruction such as this is known as an **unconditional branch** because it moves the operation directly to another instruction – there are no conditions governing this move.

## Conditional branch

Sometimes the jump can only be made if certain conditions are met. For instance, where a jump to a new sequence must not be made until a certain sequence has been executed a number of times.

This situation is presented in *figure 10a* in the form of a flow chart. The counter that is initialised in step **1**, is set to the number of times that it must go around the loop. The box in step **2** contains the actual sequence of instructions. When these are executed, step **3** decrements the counter.

The loop has been repeated the required number of times when the counter is equal to zero. The decision to either stop or repeat the loop is made at stage **4** in the diamond box. This is the **conditional branch instruction** – the conditon being set by the value of the counter.

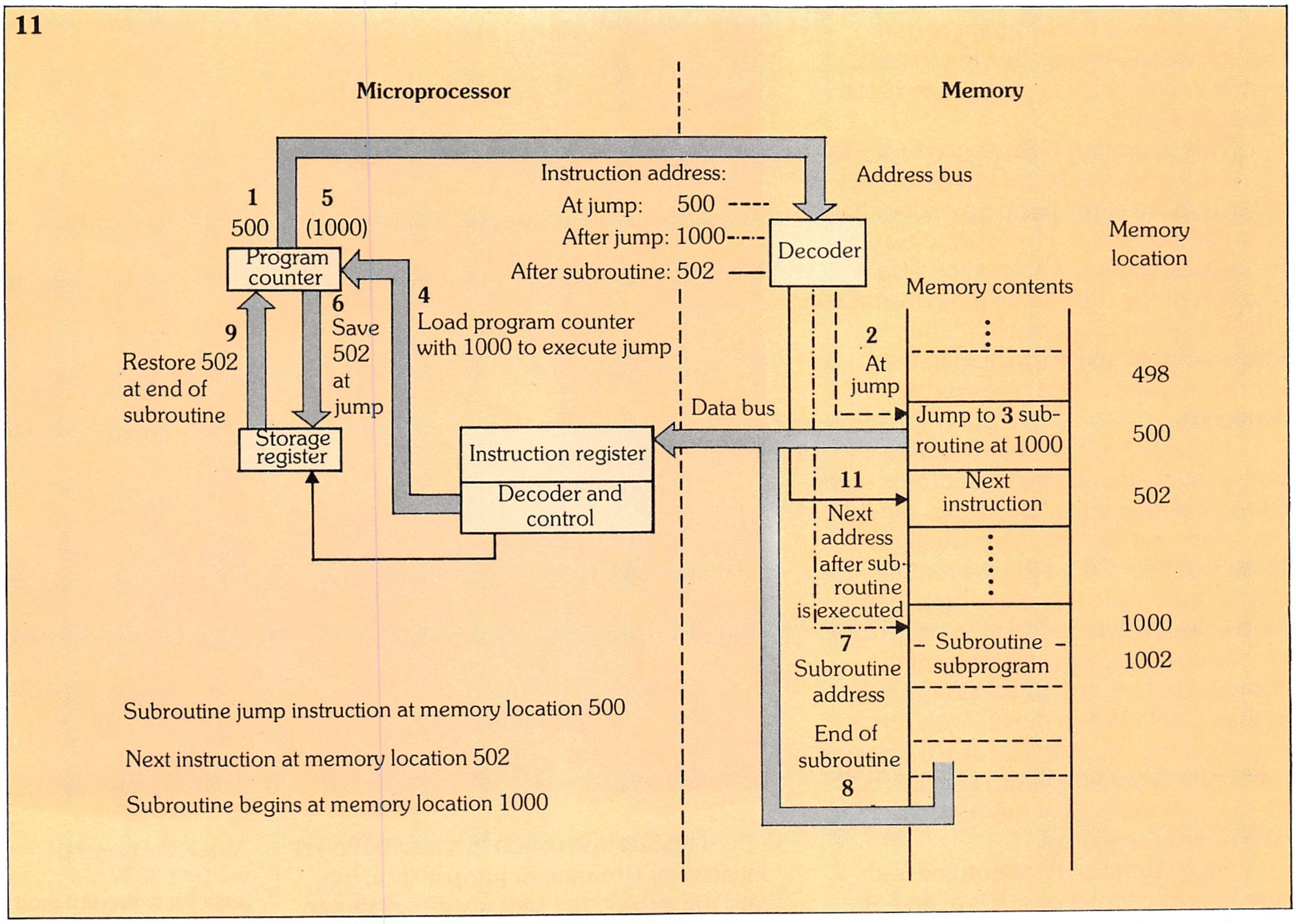
*Figure 10b* lists the different sections of an assembler program that would perform this loop eight times. The sequence begins with a data movement instruction (MVI C,8) that sets the counter's initial value to 8. As we can see, the loop instructions are not detailed, but are covered by a general notation. After this section of the program, the counter is decremented by the instruction DCR C.

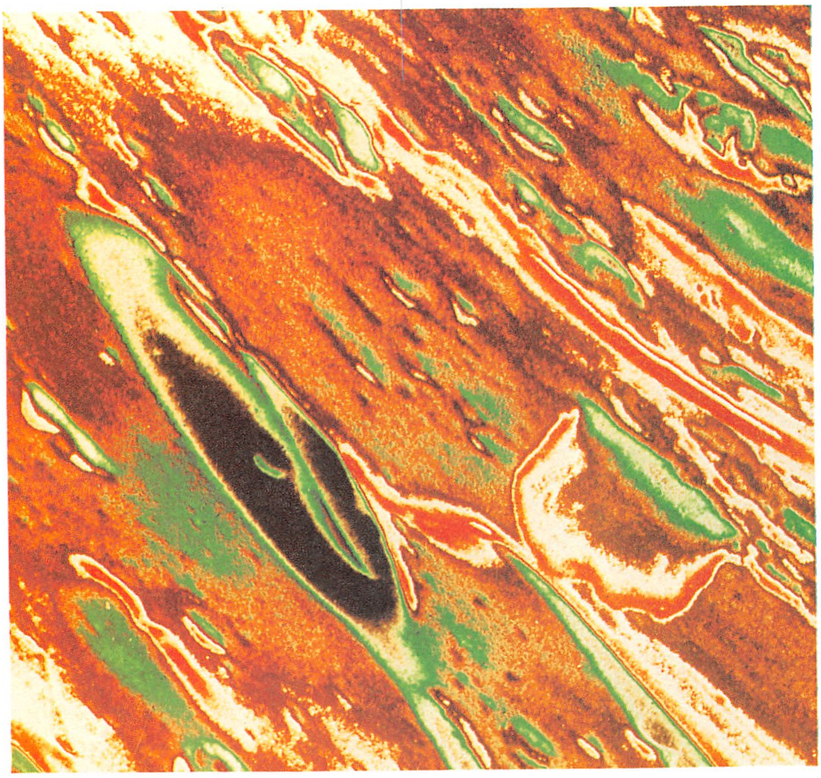
The next instruction is the conditional branch instruction, JNZ LOOP, which means 'Jump on Not Zero to the location labelled LOOP.' This is a 'branch on not zero' conditional loop instruction. The zero flip-flop is checked, and if its value is 0, the system loops back through the sequence. If the zero flip-flop is a 1 (if the register value is zero, the zero flip-flop is set to 1, remember), the system does not loop back, but goes onto the following instructions in the program.

Many other conditional jump instructions are also available. Some of these make simple checks, such as finding out if

**Right:** photomicrograph of an integrated circuit, ×5000.
(Photo: IBM).

11





the last result was positive, or if there was a carry or a borrow. Other conditional branch operations make more complicated decisions using several of the condition bits stored in the status register – such as the overflow and sign bits that were mentioned previously. This variety of conditional branch instructions means that most types of decisions can be easily written into a program.

### Linking in subroutines

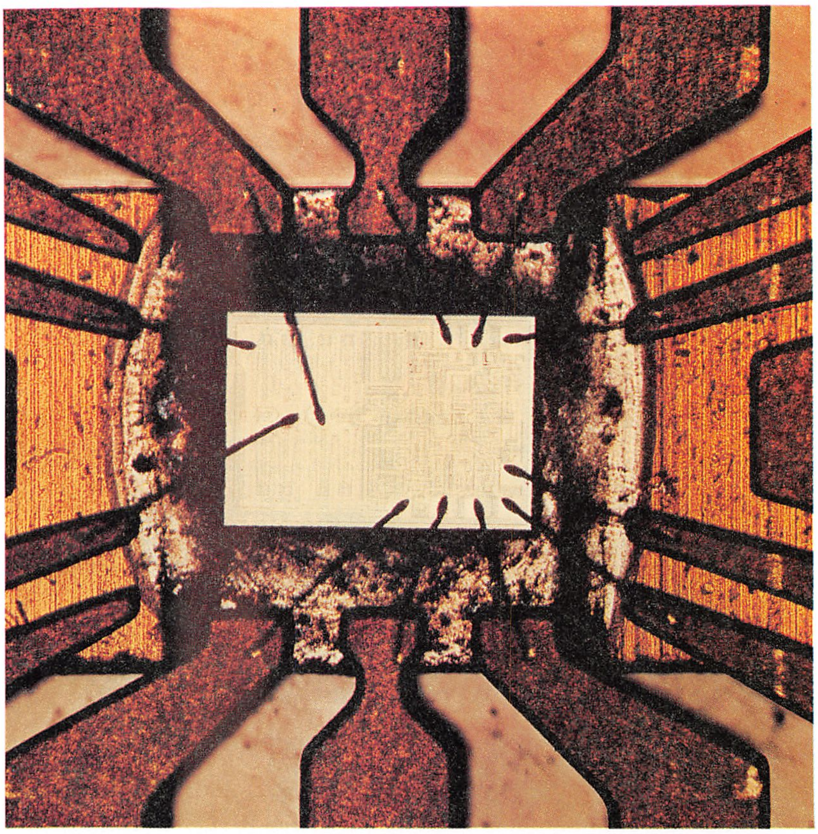
The problem with the branch instructions that we have seen so far is that they provide no mechanism for returning to the original sequence of instructions. The branch instruction that provides this link is known as a **subroutine branch** or **jump**.

The subroutine jump determines the address of the next instruction in the main sequence after the jump statement, and stores it somewhere in memory. When the subroutine is finished, a return instruction causes the saved address value to be loaded back into the program counter. This, of course, means that the system returns to where it was at the time of the subroutine jump.

This operation is illustrated in *figure 11*. As you can see, steps **1** to **5** are the same as those in the jump operation that we looked at earlier. However, because the instruction stored at location 500 is now that of the subroutine jump instruction 'JUMP to subroutine at 1000', the address of the next instruction in the normal sequence (502) is stored at location **6**.

At stage **7** the program counter directs the system to the first address of the subroutine sequence, 1000, and the system continues step by step to the end of the subroutine. When the subroutine finishes (at stage **8**), another branch or jump instruction is decoded to control the reloading of the 502 address from the storage register, back to the program counter (at **9**). When the 502 address is placed on the address bus (at **10**), the next instruction is obtained from location 502 (at 11), and the system has now reverted to its normal sequence.

So, to sum up. A subroutine is a sequence of instructions. It is a kind of subprogram that is usually designed to perform specific tasks that occur repeatedly in a program.



**Above:** this device is used in a 20 W hi-fi amplifier to prevent short circuits.
(Photo: SGS).

The first instruction in a subroutine is located by a branch or jump instruction, and the subroutine sequence contains an end instruction that returns the system to the point where the original program broke off.

## Glossary

| | |
|---|---|
| **branch or jump** | a program is said to branch when an instruction other than the next in the program sequence is executed |
| **conditional branch** | a branch that occurs if specified criteria can be met |
| **shift** | a movement of stored data, to the left or right |
| **status bit** | indicator bit, that shows the outcome of an operation – if there was a carry or if the result was negative, for example |
| **status register** | 'row' of different status bits, held in a microprocessor |

ELECTRICAL TECHNOLOGY

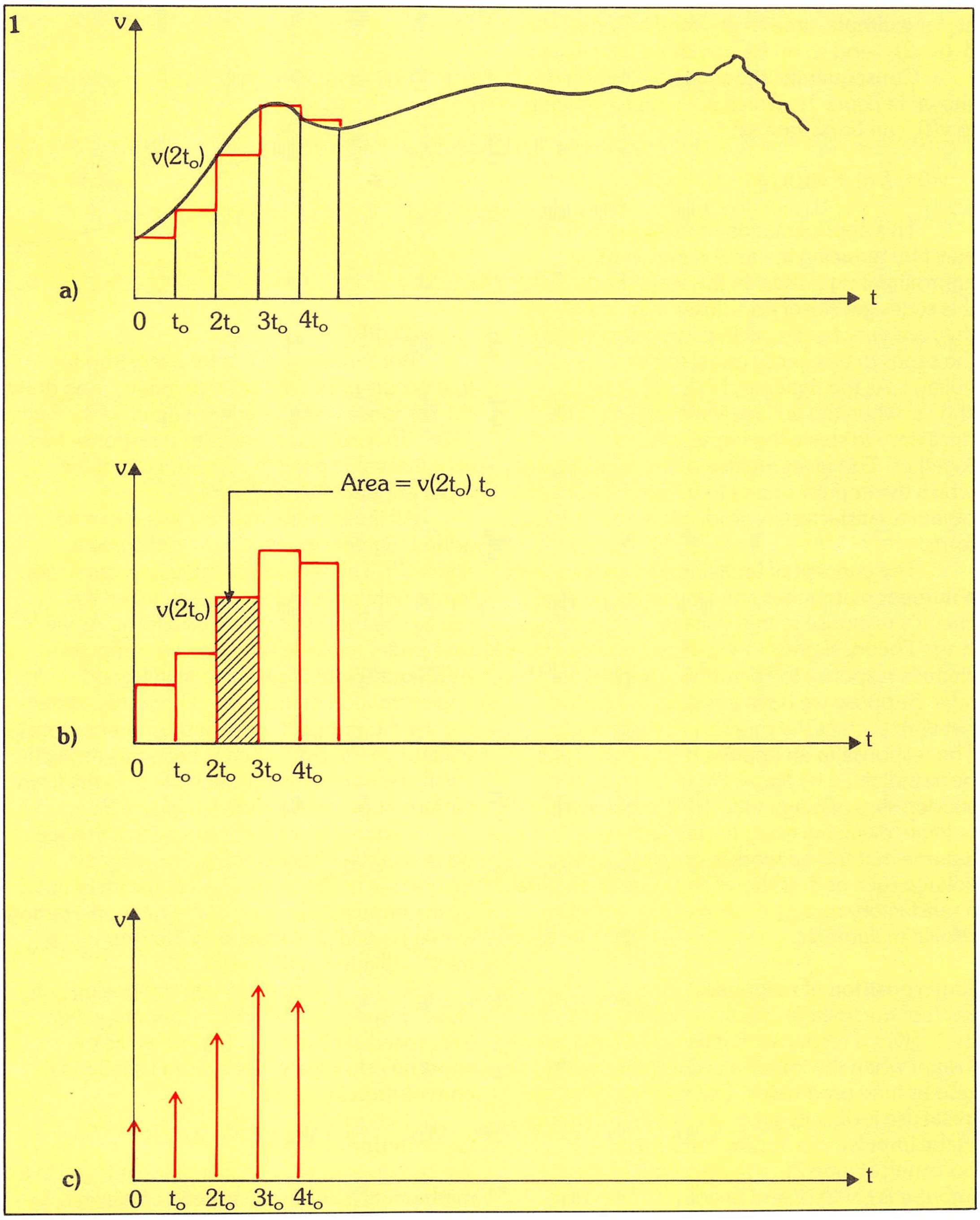
# Convolution

We'll return now to the problem outlined in the previous *Basic Theory Refresher* – how to obtain the output voltage of a linear network in response to an input voltage, which may be a relatively complex waveform.

## Superposition of impulses

*Figure 1a* shows a voltage, v(t), varying in a random manner with time. If we take a succession of time instants, 0, $t_o$, $2t_o$ ..., separated by time intervals $t_o$, at which the voltage has values, $v(0)$, $v(t_o)$, $v(2t_o)$..., the resulting sequence of voltages thus represents a crude approximation to the original wave. This is shown by the red line in *figure 1a*. This particular approximation can also be represented by the series of separate pulses illustrated in *figure 1b*. Each pulse is of duration $t_o$

**1. (a) Voltage v(t) varying** in a random manner with time; **(b)** represented by a series of separate pulses; **(c)** each pulse replaced by an impulse of voltage.



and consecutive pulses have heights: $v(0)$, $v(t_o)$, $v(2t_o)$...

So, a voltage $v(t)$ may be approximated by an infinite sequence of pulses occurring at successive instants of time: $0$, $t_o$, $2t_o$... The area contained in each of these pulses is: $v(0)t_o$, $v(t_o)t_o$, $v(2t_o)t_o$... This is shown, for the third pulse in *figure 1b,* by the shaded area.

Now let's replace each pulse with an impulse of voltage *(figure 1c).* We have seen that a unit impulse (having an area equal to one) occurring at time $t = 0$, may be written $\delta(t)$. In the same way a unit impulse occurring at, for example, time $t=2t_o$, may be written as $\delta(t-2t_o)$ and so on for any other time instant.

Consequently, the sequence of pulses shown in *figure 1c,* which is an approximation to $v(t)$, can be written as:

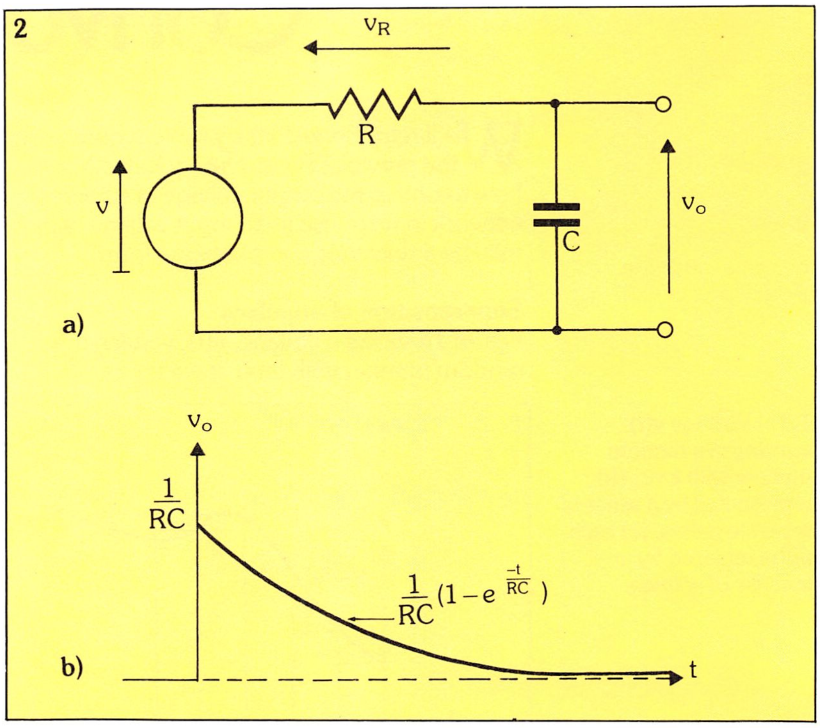
$$v(0)t_o\,\delta(t) + v(t_o)t_o\,\delta(t-t_o) + v(2t_o)t_o\,\delta(t-2t_o) + ..$$

This approximation may be made more exact by reducing the time duration of the approximating pulses. In this way, the terms of this series get closer and closer together. When they are very tightly packed, we can consider the series to be exactly equal to the original voltage. As the right hand side of *figure 1a* shows, when the voltage varies rapidly it is necessary to space the impulses closer together. This gives an idea of the frequency at which the impulses need to be taken in order to obtain a satisfactory reproduction of any waveform.

The concept of breaking a voltage up into a number of impulses can be used to analyse a circuit's response to that voltage. The previous *Basic Theory Refresher* explained how to find a circuit's response to an impulse of input voltage. Suppose we have a resistor/capacitor network such as the one shown in *figure 2a.* The response to an impulse of strength 1 will be as indicated by *figure 2b*, which shows a sudden step of magnitude 1/RC in the output voltage, decaying away to zero with time. Let's assume that this network is supplied with a voltage such as that shown in *figure 1a* and that a satisfactory representation of this voltage is shown in *figure 1c.*

### Superposition of responses

Each of the voltage impulses at times $0$, $t_o$, $2t_o$, $3t_o$... gives a response that rises suddenly at the instant when the impulse occurs and decays to zero as time progresses. The magnitude of the initial rise is directly proportional to the strength of the impulse. So, for the third pulse – occurring at time $2t_o$ – the strength of the impulse is $t_ov(2t_o)$, and the output response voltage rises suddenly *(figure 3a),* to $t_ov(2t_o)/RC$.



**2. (a) Resistor/capacitor network; (b)** graph showing response of impulse of strength 1.

We can now do this for every impulse that occurs from time $t = 0$ to infinity, and draw the response voltages at each time, as shown in *figure 3b*. Each separate output response has an initial value given by the strength of the input impulse divided by RC.

All these individual responses can be added together to give us the red curve in *figure 3b*. This is a rather crude approximation to this network's output voltage when it is excited by the given input waveform. As we said earlier, we need to record the impulses sufficiently close together to give a good approximation of the wave. Obviously, reducing the duration of $t_o$ gives a smoother output voltage. In the limit when $t_o$ becomes negligibly small, we obtain the output voltage in the form of the smooth curve shown in *figure 3c*.

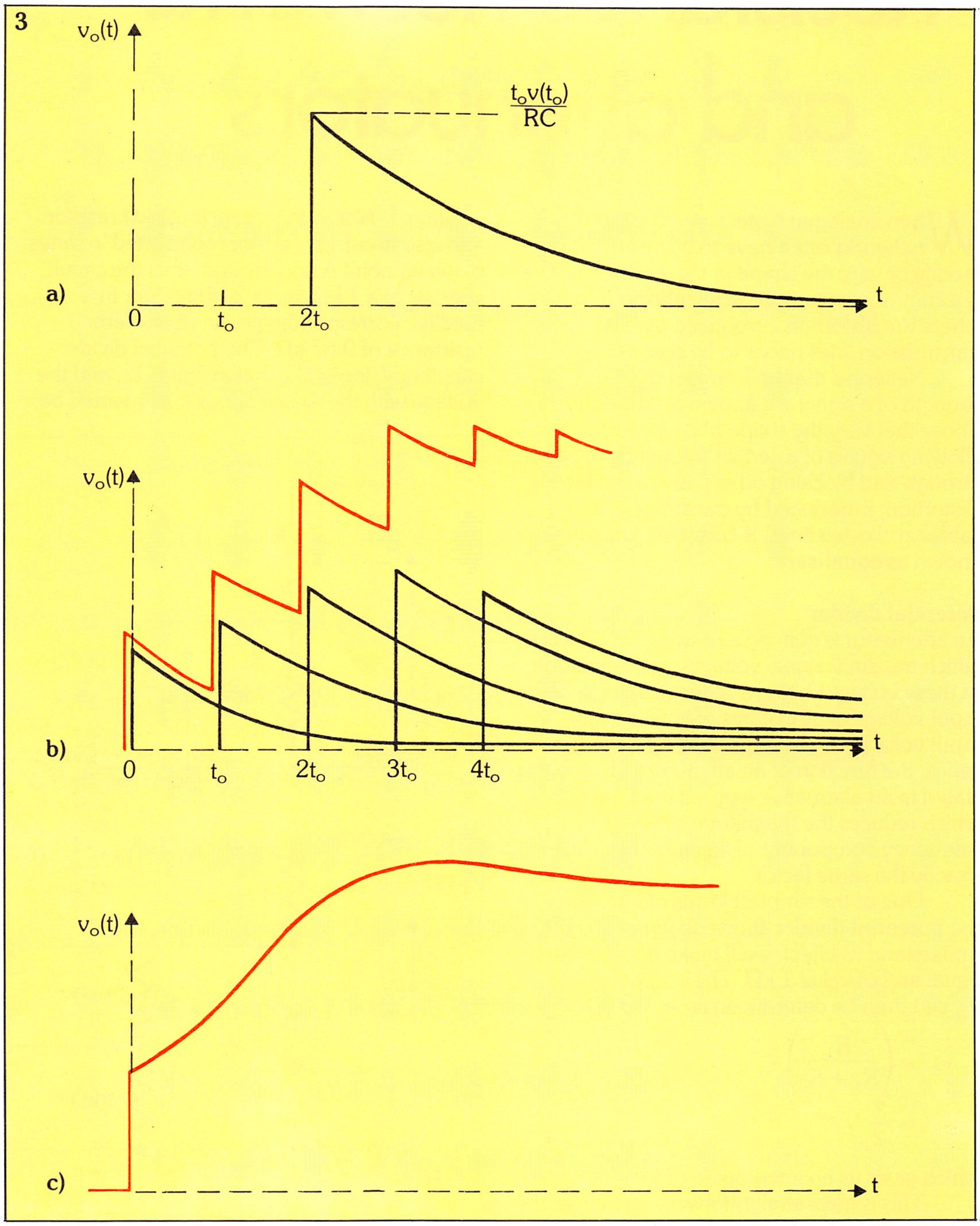
Thus a network's response to a voltage wave may be obtained by multiplying the magnitude of the wave at each instant of time, by the impulse response of the network starting at that particular instant, and then summing these individual responses.

As this is derived by superimposing individual responses and adding them together, the procedure is sometimes known as the **superposition integral** or more usually as the **convolution integral.**

### Convolution integral

We can now write the convolution integral in a mathematical form but, instead of taking $t_o$ to be infinitesimally small, we'll assume it to be

**3. (a) Each voltage impulse** gives a response that rises suddenly at the instant that the impulse occurs and then decays to zero; **(b)** response voltages at each time instant; **(c)** adding individual responses gives this curve.



finite in size, but still very tiny.

Let us look at our network's response to a unit impulse at time t = 0, and write it as h(t). This is usually called the **impulse response** of the network. If the unit impulse is applied at a time t = T, then the response to an impulse may be written as h(t – T).

If we now consider an input voltage v(t), starting at t = 0, and convert this to impulses at times separated by short intervals $t_o$, then the output voltage, $v_o$, due to the separate impulses can be written as follows: at t = 0, the response to the first impulse is $t_o v(0)h(t)$; the response to the impulse at t = $t_o$ is $t_o v(t_o)\ h(t-t_o)$; to the impulse at t = $2t_o$, the response is $t_o v(2t_o)\ h(t-2t_o)$; and so on.

Adding these together gives the output voltage $v_o(t)$ as:

$$v_o(t) = t_o\,[v(0)h(t) + v(t_o)\,h(t-t_o) + v(2t_o)\,h(t-2t_o) + \ldots]$$

The series is continued to include all the terms needed for our problem. In practice, $t_o$ is made as small as is necessary to obtain an accurate answer, but not so small that too many terms need to be introduced in the addition. □

ELECTRICAL TECHNOLOGY

# Resistance networks and attenuators

When analogue systems are designed, networks often have to be inserted to modify or vary the shape of the waveform. This is a common occurence in telephone systems where the distortion introduced by long transmission lines needs to be corrected.

Networks that uniformly reduce the strength of a signal are known as **attenuators**. Those that vary the shape of a waveform, by allowing signals of a certain frequency to pass through and blocking others, are called **filters**. Distortion, introduced by transmission along cables and other lines, is corrected by networks known as **equalisers**.

## Potential divider

An attenuator is defined as a two-port network which transmits signal voltages; the amplitude of these voltages being a reduced version of the input voltage at all instants of time. As any input voltage can be expressed by a Fourier series, the function of an attenuator may be stated in an alternative way – it is a network which reduces the magnitude of every frequency component of an input wave by exactly the same factor.

One of the simplest forms of attenuator is the **potential divider** shown in *figure 1a*. To understand its effect, we'll make both resistors equal and of value 1 kΩ. The output voltage, $v_1$, can then be determined as:

$$v_1 = \left(\frac{R_1}{R_1 + R_2}\right) v = \frac{v}{2}$$

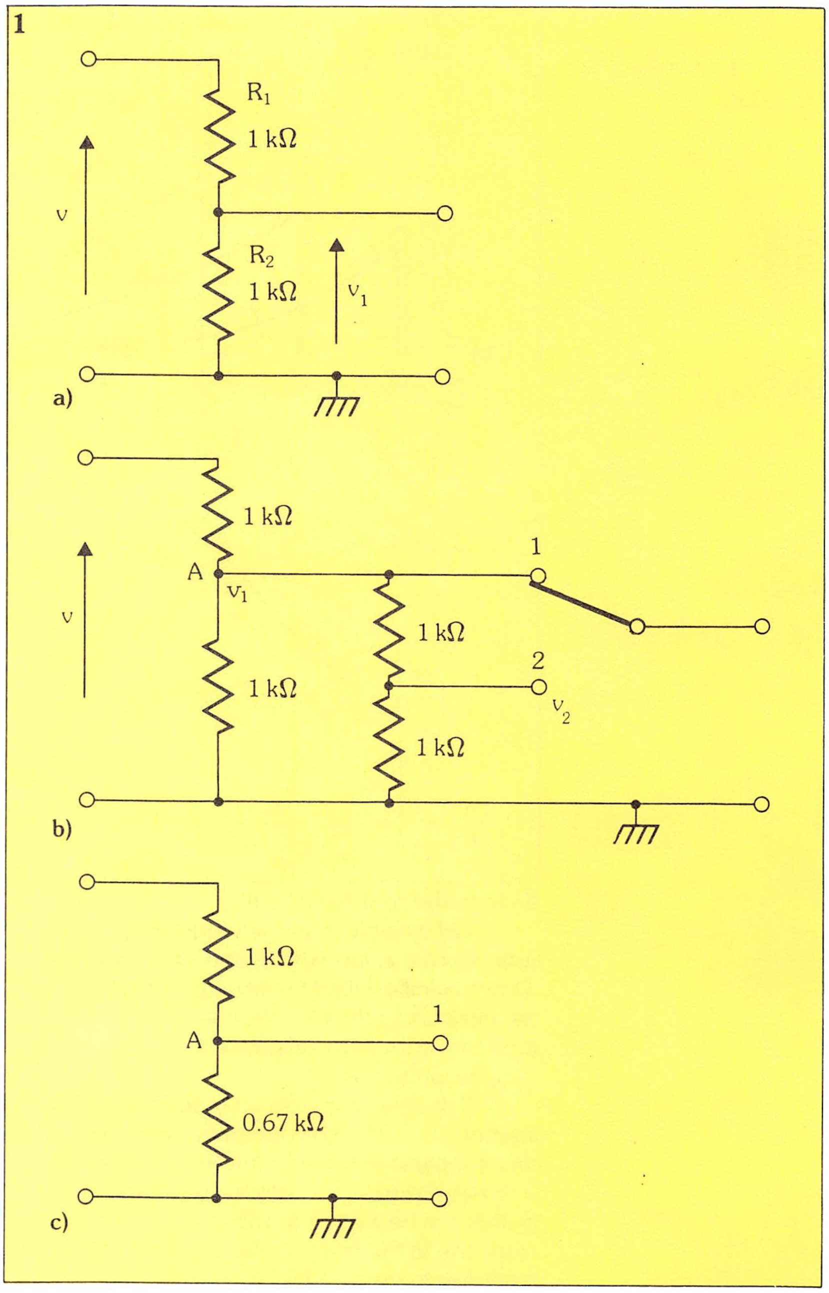
which gives us an attenuation of ½.

This is quite straightforward so long as the network which is connected at the output does not take any current from the input. (In other words, it must have an infinitely large input resistance.)

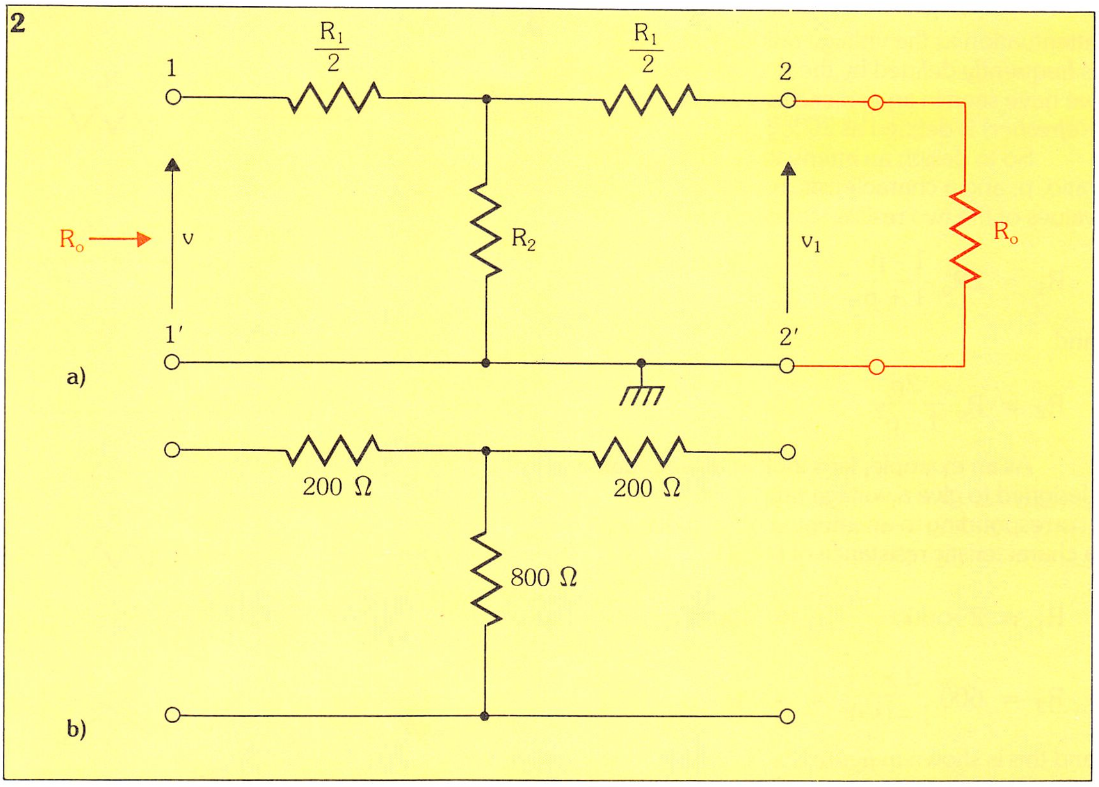
Suppose that we need to design an attenuator which gives an attenuation of either ½ or ¼, depending on the position of a switch. We might think that a second potential divider could be connected across the first as shown in *figure 1b*. This provides the choice of either voltage $v_1$ or voltage $v_2$ as the output, to give v/2 and v/4 respectively. However, this does not work.

Look at the circuit with the switch in position 1. Notice that there is a 1 kΩ resistor and also two 1 kΩ resistors connected in series between point A and ground: thus the circuit consists of a 1 kΩ and a 2 kΩ resistor in parallel, corresponding to an equivalent resistance of 0.67 kΩ. The potential divider circuit now looks like that in *figure 1c*, and the voltage with the switch at position 1 would be:

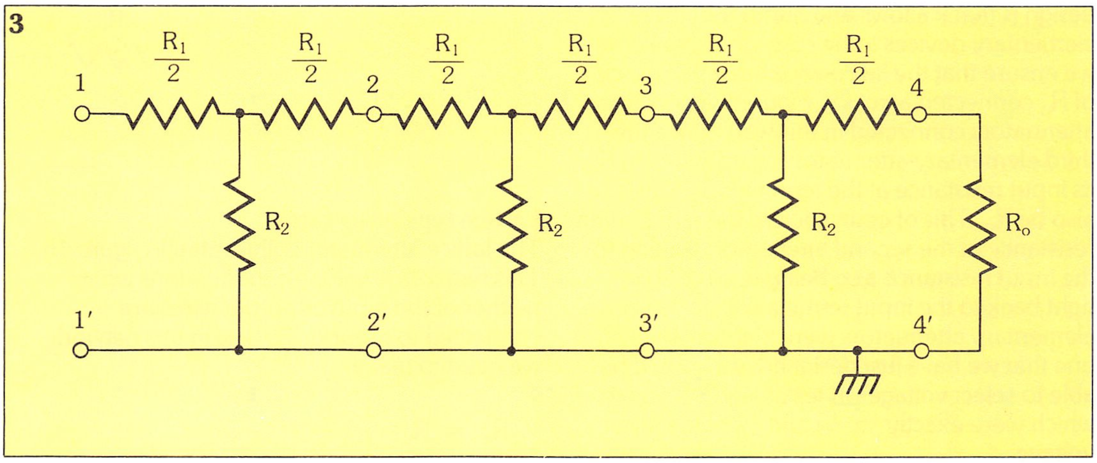
**1. (a) A potential divider** – the simplest type of attenuator; **(b)** a second potential divider connected across the first; **(c)** equivalent circuit of that in **(b)**.



**2. T type attenuators** are unaffected by others in cascade.



**3. Cascaded T type** attenuator.



$$\frac{0.67}{1 + 0.67} v = 0.4\,v \quad \text{(i.e. } \textit{not}\ 0.5\,v\text{)}$$

Similarly with the switch at point 2, the voltage will be 0.2 V instead of the required 0.25 V. This is because the desired attenuation is only obtained when the output terminals are not loaded by a following circuit.

### T type attenuator

This difficulty can be overcome by designing an attenuator which is not affected when another, or even a succession of similar attenuators, is cascaded with it. The basic element is shown in *figure 2a* and consists of two series elements, each of value $R_1/2$, and a shunt element of value $R_2$, arranged in the form of the letter T.

This network has to be designed so that when a resistor of value $R_o$ is connected across the output terminals, 2, 2′, the resistance seen looking into the input terminals, 1, 1′, is also $R_o$.

With the resistance $R_o$ connected as shown, the resistance looking in terminals 1, 1′ is:

$$\frac{R_1}{2} + \frac{R_2\,(R_o + R_1/2)}{R_2 + R_o + R_1/2}$$

which must be equal to $R_o$. This leads us to a value for $R_o$:

$$R_o = \tfrac{1}{2}\sqrt{R_1\,(R_1 + 4R_2)}$$

In addition to this, we may define the

attentuation as the voltage ratio $p = v_1/v$. This is frequently defined by the decibel which (as we have seen in an earlier *Basic Theory Refresher*) is defined as $20 \log_{10} p$.

So to design an attenuator with a voltage ratio, p, and a characteristic resistance, $R_o$, the values of the two resistances need to be:

$$R_1 = 2R_o \frac{1-p}{1+p}$$

and:

$$R_2 = R_o \frac{2p}{1-p^2}$$

As an example, let's look at an attenuator designed to give a voltage ratio of ½ (corresponding to an attenuation of 6 dB) with a characteristic resistance of 600 Ω:

$$R_1 = 2 \times 600 \frac{1-½}{1+½} = 400\ \Omega$$

$$R_2 = 600 \frac{2 \times ½}{1-(½)^2} = 800\ \Omega$$

and this is shown in *figure 2b*.

The great advantage of this attenuator design is that it allows any number of these elementary devices to be cascaded, providing we ensure that the last device has a resistance of $R_o$ connected across it. *Figure 3* shows three attenuators connected in this way. Since the third elementary attenuator is terminated in $R_o$, its input resistance at the terminals 3, 3′ will also be $R_o$. This of course forms the terminating resistance of the second attenuator, leading to the input resistance also being $R_o$, and so on right back to the input terminals. If all these elementary attenuators were identical to the one that we have just designed, we should be able to select voltages at terminals 2, 3 and 4 which were exactly ½, ¼ and ⅛ of the input voltage.

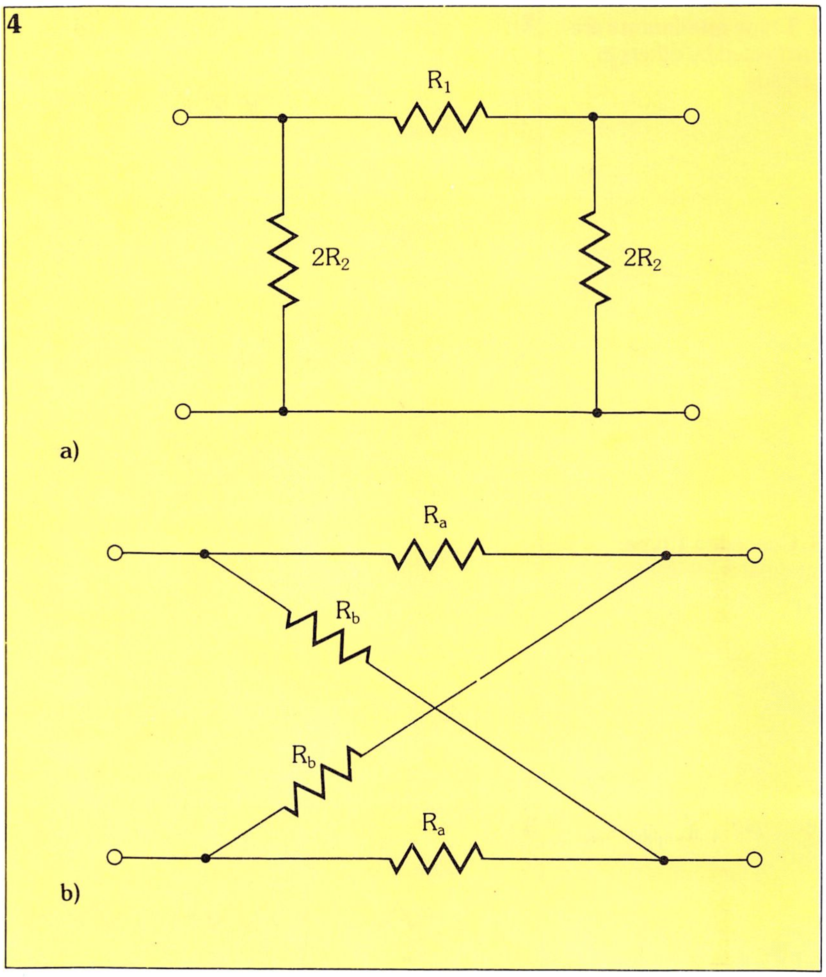
It is easy to see that the attenuation of each of the networks does not have to be identical, so long as they are all made to have the same characteristic resistance. This gives us even greater flexibility in design.

### π type attenuator

A slightly different type of basic network is shown in *figure 4a*. This comprises a circuit connected in the shape of the greek letter π. Designing this network to give a voltage ratio p, and characteristic resistance $R_o$, gives us:

$$R_1 = R_o \frac{1-p^2}{2p}$$

$$R_2 = \frac{R_o(1+p)}{2(1-p)}$$



4. (a) π type attenuator; (b) lattice type attenuator.

### Lattice type attenuator

The lattice attenuator is illustrated in *figure 4b*. This network is valuable in situations where neither of the input or output wires are connected to ground. To design this network we need to make:

$$R_a = R_o \frac{1-p}{1+p}$$

$$R_b = R_o \frac{1+p}{1-p}$$

□

MICROPROCESSORS 6

# Locating information

## Addressing modes

So far, we have seen that program counters provide addresses for instructions, and that these instructions provide addresses for other instructions and for data held in memory. To find out more about this process we need to discuss **addressing modes**. These are the permitted ways that are used to locate information. The most common addressing modes used by microprocessors are:

1) immediate addressing;
2) register addressing;
3) register indirect addressing;
4) indexed addressing;
5) direct addressing;
6) relative addressing.

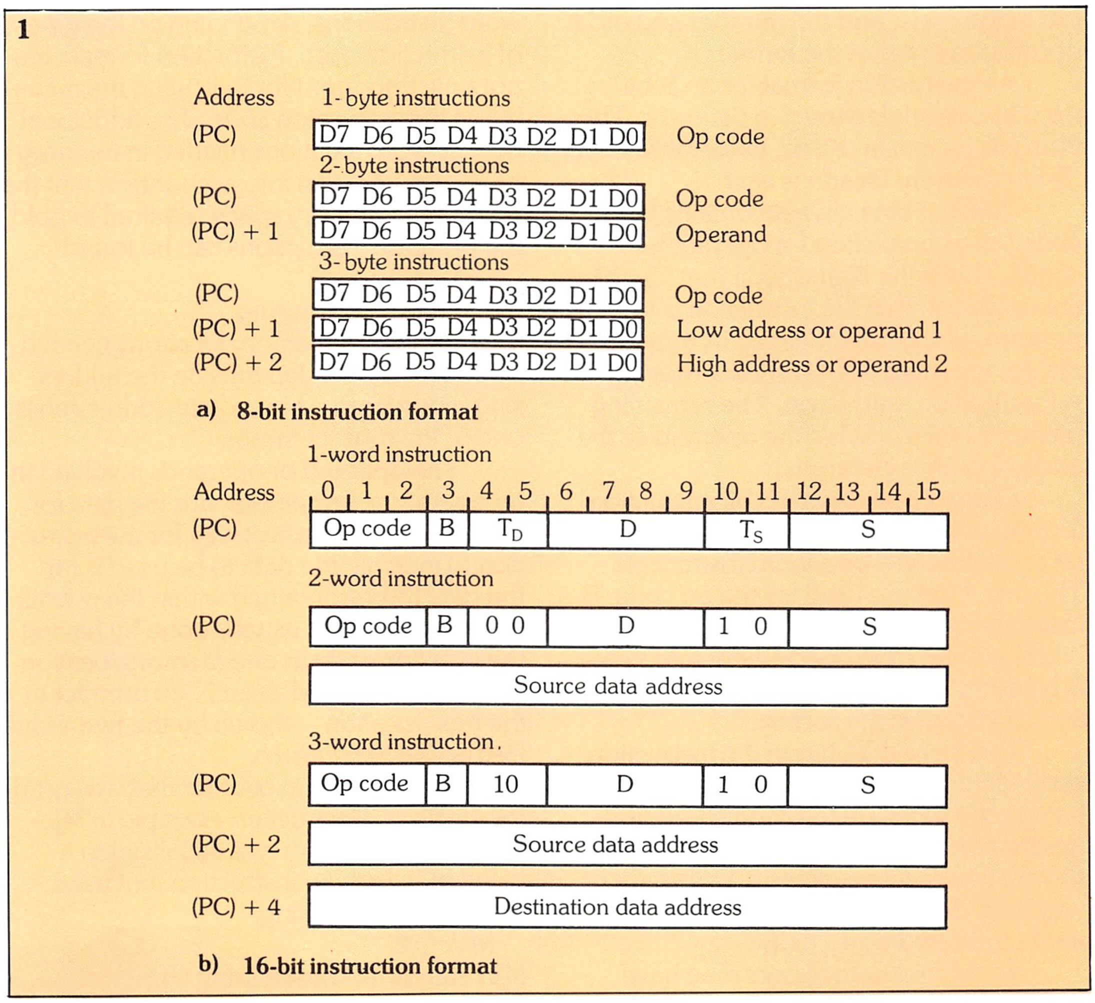
Not all microprocessors use all these modes, but most are fairly common.

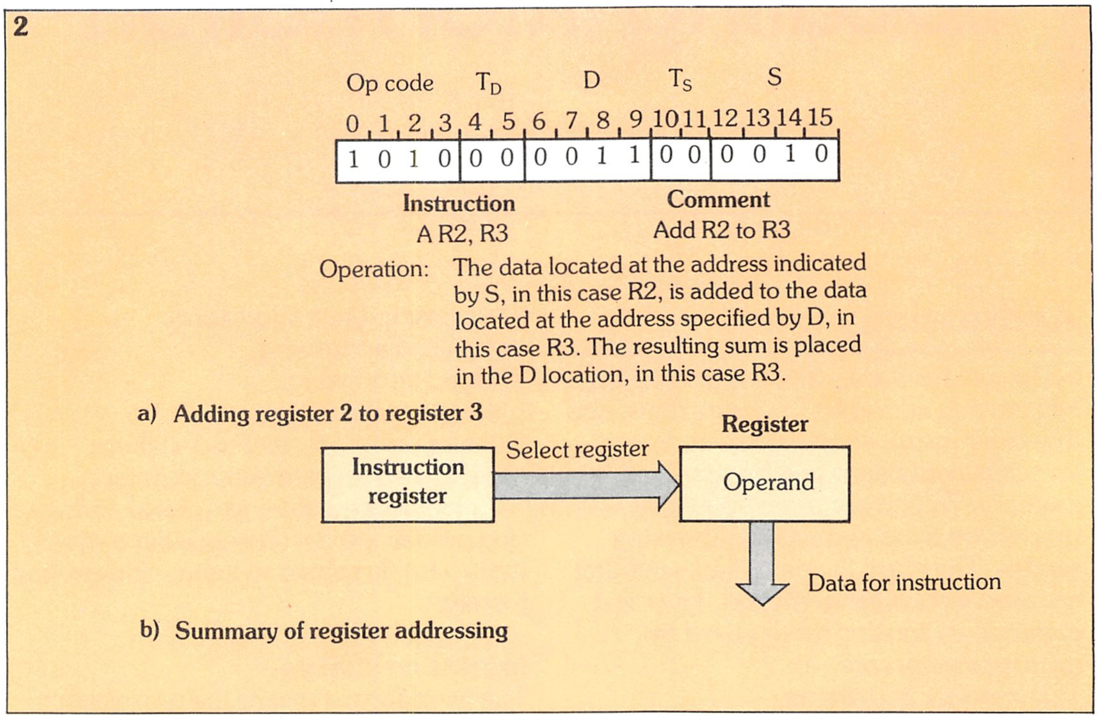
However, before we look at addressing modes, we need to consider a subject that is closely related to them – **instruction formats**.

### Instruction formats

The word format means the organisation of an item. In the computer, instructions

**1. Instruction formats for (a)** 8-bit; and **(b)** 16-bit microporcessors.





Operation: The data located at the address indicated by S, in this case R2, is added to the data located at the address specified by D, in this case R3. The resulting sum is placed in the D location, in this case R3.

a) **Adding register 2 to register 3**

b) **Summary of register addressing**

**2. Register addressing.**

take the form of digital codes – the order of bits in the code, and the number and order of codes are part of the format.

An instruction format for an 8-bit microprocessor is shown in *figure 1a*. The instructions can take one, two or three byte (8-bit) memory locations each.

The first byte always contains the **op code** (an abbreviation for operations code). This is the digital code that identifies the operation that the instruction is to perform, and corresponds to its mnemonic, for example, MOV for data movement, A for addition, and so on. The remaining memory bytes provide the operand or the address of the operand(s).

*Figure 1b* shows a 16-bit instruction format, which allows more information to be contained in a one-word instruction. The first 4 bits are used for the op code. B instructs the microprocessor to use all 16 bits as a word or to divide them into two, 8-bit bytes. The code identifies the destination; S identifies the source.

The $T_D$ and $T_S$ bits in the instruction code identify the kind of addressing mode being used to locate the source and the destination of the instruction. As *figure 1b* shows, two or three memory words are required, depending on the addressing mode indicated by $T_S$ or $T_D$.

Some microprocessors may have more than one instruction format for one word instructions, depending on the *type* of instruction used. Instruction formats are not only important for indicating the meaning of the instruction code. The additional byte or word locations needed in memory must also be taken into account, so that the amount of memory space required to hold the program instructions can be found.

### Immediate addressing

The number of memory locations needed by an instruction depends on the addressing mode used – immediate addressing is one of the most common.

The operand or operands involved in an operation, remember, are the data for processing. The easiest way for the instruction to indicate the data to be used is for the data to be contained within the overall instruction. This is usually done by having the N-bit op code in one memory location and the data stored as an N-bit number in the next location – shown by the two-word instruction in *figure 1b*.

Recall the loop counter that was used for the JUMP instruction example in *Microprocessors 5*. This was initialised to a value of 8, with an instruction that used immediate addressing:

MVI C,8

MVI stands for 'move using immediate

addressing'; C identifies the counter register to be initialised; and 8 represents the value to be stored in the C register. The 8-bit op code for MVI C is stored in the first 8-bit byte of the instruction, and the binary equivalent of the decimal number 8 is stored in the second 8-bit byte. The instruction thus needs a total of 16 bits (two, 8-bit bytes) which take two memory locations.

### Register addressing

The MVI C,8 instruction is also an example of register addressing. The instruction says that the register C is loaded with the constant 8. The register C has thus been specified as one of the data locations involved in the operation.

This type of addressing is often used in programming as, once the register involved is identified, the operation can take place immediately, without the need to go to memory for more data address information.

A more specific example is shown in *figure 2a*. The instruction code contains all the information necessary to add register 2 to register 3, and to store the result in register 3. The instruction is executed as soon as it is decoded, since all the data locations are held within the processor. No additional memory references are needed.

The $T_D$ and $T_S$ bit-pairs are used to indicate the addressing mode. The 00 in each of these bit fields for this format indicate that register addressing is used for both the source and destination. The op code is 1010, which identifies the instruction to the microprocessor as an APP operation. Register addressing is summarised in *figure 2b*.

### Register indirect addressing

With this type of addressing, the address of data in memory is contained in one of the microprocessor's registers. This is a different operation to register addressing, as the contents of the register are now the *address* of the data, rather than the data itself. All the instruction has to do is indicate which register contains the address to be used.

The instruction from the instruction register is decoded, and the register that holds the data address is selected. The data address is then sent to memory to locate the data, which in turn is sent back to the microprocessor on the data bus. While time must be taken to read memory to fetch the data, or to write the data into its memory location, the instruction coding can be very simple.

Generally speaking, the N-bit instruction op code, needed for an N-bit processor, contains all the necessary information. Indirect addressing is very useful where an application demands that data has to be accessed from memory in some sequence. The same instruction is used for each data item, but the register value is incremented by one, to address the next memory location each time. Changes can easily be made to the data address by loading new values or performing some arithmetic operation on the address register.

### Indexed addressing

Indexed addressing – a variation on indirect addressing that is used by some microprocessors – is illustrated in *figure 3*. In this mode, the operand or data address is held in two locations. There is a **base value** in the index register, and an **offset value** in the instruction code. When the instruction is executed, the offset value is added to the index register value to determine the final data address.

Using this procedure, the address can be modified by operating on the contents of the index register. Data that is spaced throughout the memory in some regular manner can be accessed by using the proper value in the instruction.

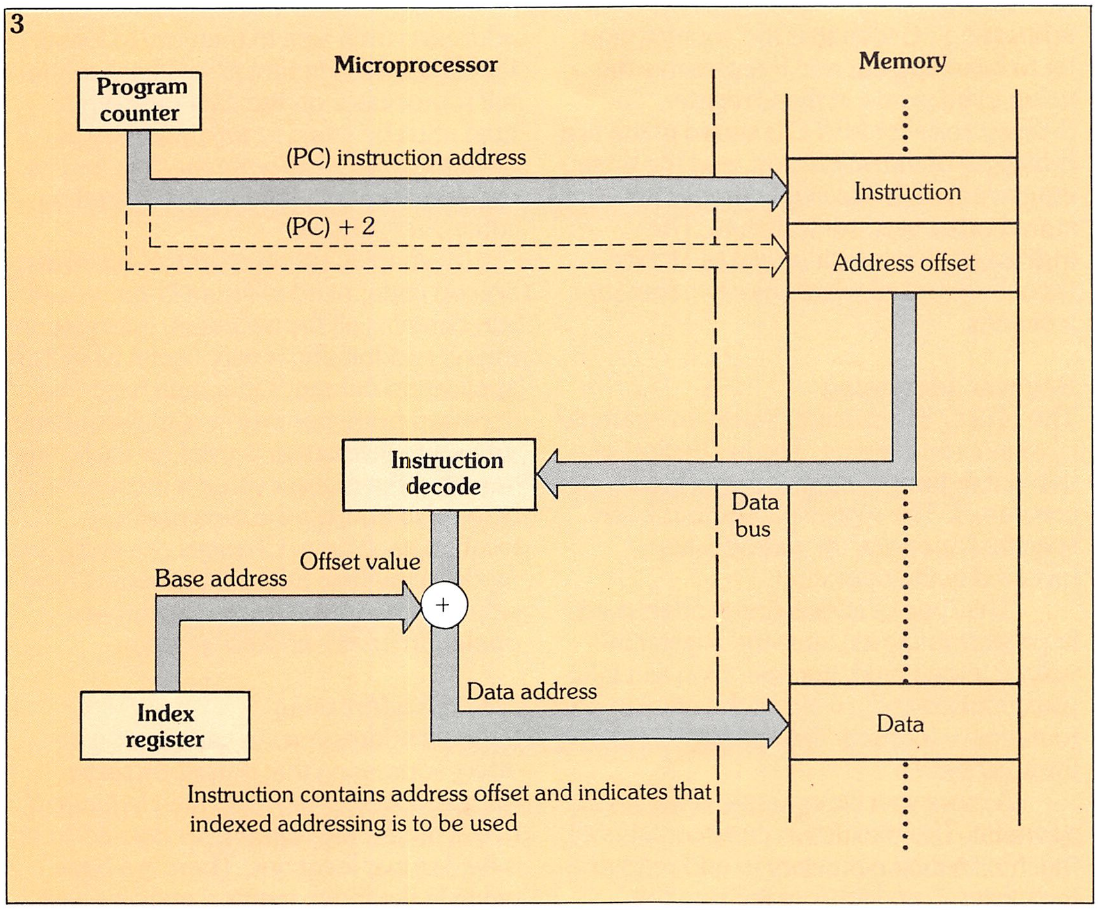
With indexed addressing, the instruction must contain two code groups. The first is the instruction operation code group, which indicates the operation, and the fact that indexed addressing is being used. The second group of bits represents the address offset.

Indexed addressing takes longer than register indirect addressing, because the program memory is read twice.

### Direct addressing

Returning to *figure 1*, we see that in the 3-byte instruction format for the 8-bit microprocessor, and in the 2-word and 3-word formats for the 16-bit microprocessor, the byte or word after the op

3



**3. Indexed addressing** is a variation of indirect addressing where the operand is held in two locations.

code contains a data address rather than data. Unlike the 2-byte instruction used for immediate addressing, where the second byte contained the data itself, the second byte (and perhaps the third, too) now contains the address of the data. This is known as **direct addressing** (also known as symbolic addressing).

Direct addressing is often used when a microprocessor register is not available for data storage. *Figure 4* shows direct addressing in operation. The program counter addresses the program memory to obtain the first part of the instruction. This contains the op code and a code that says direct addressing is being used. This, in turn, directs the program counter to be incremented so that the program memory can be read again to obtain the data address, or addresses. Once the location of the data has been determined, it can then be read in from RAM.

This type of addressing is very slow as it can require four memory operations to get the data location. It is usually only used when register addressing is impossible or when a single data variable – such as a program counter or control word – is only used once in a while.

### Relative addressing

All of the addressing modes that we have looked at so far have been ways of locating data to be used by the instruction under execution. However, instructions themselves are sometimes addressed in a program, and therefore have to be located. For example, branch and jump instructions.

Relative addressing is often used in branch operations to specify where the next instruction is located (the instruction to branch to). This addressing method has some similarities to indexed addressing, in that the instruction contains an offset number that is added. The difference here is that the base address is the value in the program counter.

The sequence of operations for relative addressing is as follows.

1) The program counter addresses the next instruction, and the instruction that contains the op code is read.

2) The microprocessor then decodes the instruction and finds that program counter relative addressing is being used.
3) The program counter is then incremented and the next location in program memory is read. This gives the binary number that represents the address offset.
4) The offset value is then added to the contents of the program counter and the result placed *in* the program counter. The new program counter value is the address of the next instruction branched to by the branch instruction.

The advantage of locating the next instruction branch address in this way is that it is faster than direct addressing. The disadvantage is that the offset value is limited. An 8-bit two's complement offset value is limited to values in the range –128 to +127. As a result, a jump or offset from the present program counter value is also similarly limited. If the next instruction branch address is not in this range, then some other (slower) addressing method has to be used. In fact programmers are constantly faced with this problem – to find which addressing procedure is best for each instruction in the program.

## Turning instructions into programs

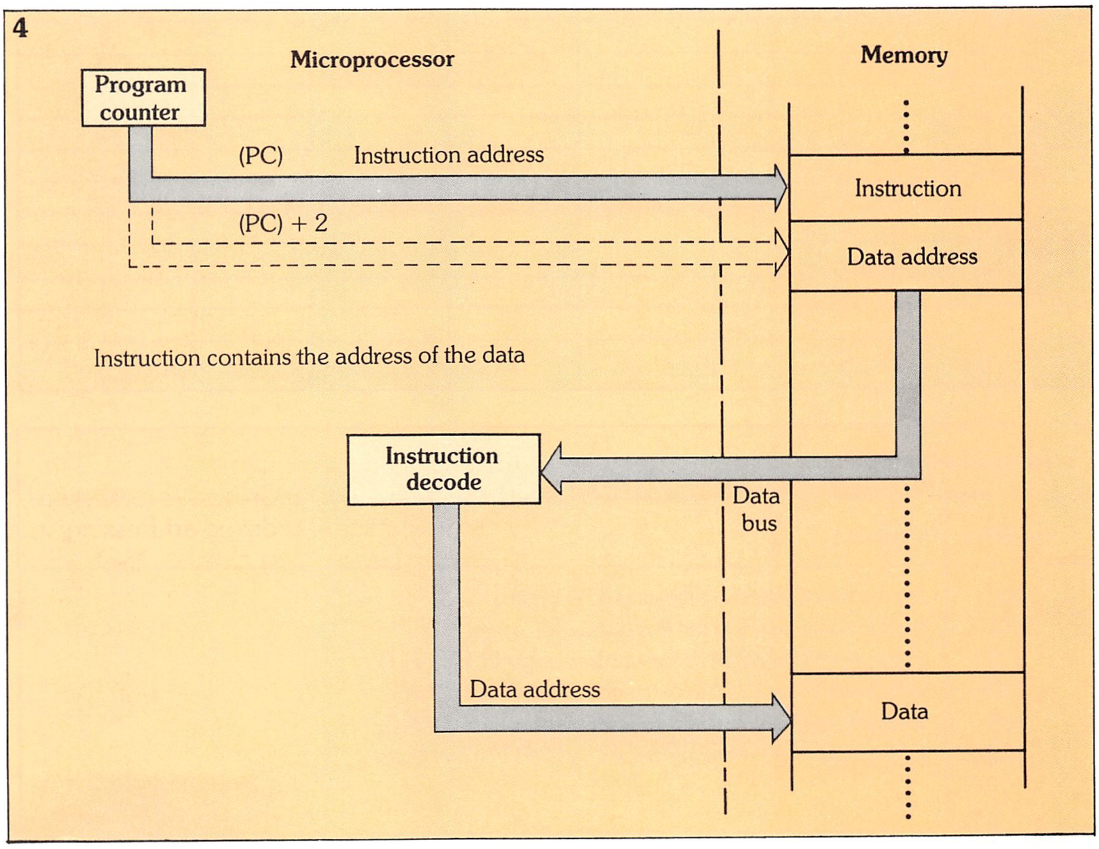
Having now looked at the way in which microprocessors function, further chapters in this series will investigate the way in which microprocessors are utilised as the heart of a programmable system. The use of instructions and the manner in which they are combined into programs is obviously of prime importance.

The overall system should be analysed and broken down into flow charts. The instruction sequences or subprograms to implement the elementary operations should then be written, verified and then combined to make the full program. The overall program should then be verified to ensure that all the subprograms are linked into a harmonious structure.

### Steps in system development

1) The first step in building any system is to completely summarize the operation of the system in tabular or flow chart form.
2) From the system flow chart, decide which microprocessor and other compo-

4. Direct or symbolic addressing.



| Step | Program counter | Data address register 1 | Data address register 2 | Data address register 3 | B register | A register | Flip-flop | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | EQ | CY |
| 1 | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | | | | | | | | |
| 3 | | | | | | | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | | | | | | | |
| 7 | | | | | | | | |
| 8 | | | | | | | | |
| 9 | | | | | | | | |
| 10 | | | | | | | | |
| 11 | | | | | | | | |
| 12 | | | | | | | | |
| 13 | | | | | | | | |
| 14 | | | | | | | | |
| 15 | | | | | | | | |
| 16 | | | | | | | | |
| 17 | | | | | | | | |
| 18 | | | | | | | | |
| 19 | | | | | | | | |
| 20 | | | | | | | | |
| 21 | | | | | | | | |
| 22 | | | | | | | | |
| 23 | | | | | | | | |
| 24 | | | | | | | | |
| 25 | | | | | | | | |
| 26 | | | | | | | | |
| 27 | | | | | | | | |
| 28 | | | | | | | | |
| 29 | | | | | | | | |
| 30 | | | | | | | | |
| 31 | | | | | | | | |
| 32 | | | | | | | | |
| 33 | | | | | | | | |
| 34 | | | | | | | | |
| 35 | | | | | | | | |

**Data memory**

Address

200
201
202
203
204

210
211
212
213
214

220
221
222
223
224

**Microprocessor**

Program counter

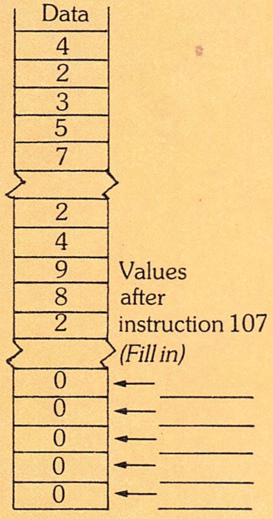
Data address register 1

Data address register 2

Data address register 3

**Program memory**

| Address | Contents |
|---|---|
| 100 | Instruction: Clear carry flip-flop (CY) to zero |
| 101 | Instruction: Place 5 in the B register |
| 102 | Instruction: Place 204 in data address register 1 (DAR 1) |
| 103 | Instruction: Place 214 in data address register 2 (DAR 2) |
| 104 | Instruction: Place 224 in data address register 3 (DAR 3) |
| 105 | Instruction: Move data addressed by DAR 1 to A register |



A register

B register

Carry flip-flop CY

Equal flip-flop EQ

| Step | Program counter | Data address register 1 | Data address register 2 | Data address register 3 | B register | A register | Flip-flop | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | EQ | CY |
| 1 | 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 101 | 0 | 0 | 0 | **5** | 0 | 0 | 0 |
| 3 | 102 | **204** | 0 | 0 | 5 | 0 | 0 | 0 |
| 4 | 103 | 204 | **214** | 0 | 5 | 0 | 0 | 0 |
| 5 | 104 | 204 | 214 | **224** | 5 | 0 | 0 | 0 |
| 6 | 105 | 204 | 214 | 224 | 5 | **7** | 0 | 0 |
| 7 | 106 | 204 | 214 | 224 | 5 | **9** | 0 | 0 |
| 8 | 107 | 204 | 214 | 224 | 5 | 9 | 0 | 0 |
| 9 | 108 | **203** | **213** | **223** | **4** | 9 | 0 | 0 |
| 10 | 109 | 203 | 213 | 223 | 4 | 9 | 0 | 0 |
| 11 | 105 | 203 | 213 | 223 | 4 | **5** | 0 | 0 |
| 12 | 106 | 203 | 213 | 223 | 4 | **3** | 0 | **1** |
| 13 | 107 | 203 | 213 | 223 | 4 | 3 | 0 | 1 |
| 14 | 108 | **202** | **212** | **222** | **3** | 3 | 0 | 1 |
| 15 | 109 | 202 | 212 | 222 | 3 | 3 | 0 | 1 |
| 16 | 105 | 202 | 212 | 222 | 3 | **3** | 0 | 1 |
| 17 | 106 | 202 | 212 | 222 | 3 | **3** | 0 | **1** |
| 18 | 107 | 202 | 212 | 222 | 3 | 3 | 0 | 1 |
| 19 | 108 | **201** | **211** | **221** | **2** | 3 | 0 | 1 |
| 20 | 109 | 201 | 211 | 221 | 2 | 3 | 0 | 1 |
| 21 | 105 | 201 | 211 | 221 | 2 | **2** | 0 | 1 |
| 22 | 106 | 201 | 211 | 221 | 2 | **7** | 0 | **0** |
| 23 | 107 | 201 | 211 | 221 | 2 | 7 | 0 | 0 |
| 24 | 108 | **200** | **210** | **220** | **1** | 7 | 0 | 0 |
| 25 | 109 | 200 | 210 | 220 | 1 | 7 | 0 | 0 |
| 26 | 105 | 200 | 210 | 220 | 1 | **4** | 0 | 0 |
| 27 | 106 | 200 | 210 | 220 | 1 | **6** | 0 | 0 |
| 28 | 107 | 200 | 210 | 220 | 1 | 6 | 0 | 0 |
| 29 | 108 | **199** | **209** | **219** | **0** | 6 | **1** | 0 |
| 30 | 109 | 199 | 209 | 219 | 0 | 6 | 1 | 0 |

**Numbers stored in memory locations at the end of the program**

220: 6, 221: 7, 222: 3, 223: 3, 224: 9

**Program memory (continued)**

| Address | Contents |
|---|---|
| 106 | Instruction: Add carry and data addressed by DAR 2 to data in A register. Set the carry flip-flop with any carry generated. |
| 107 | Instruction: Move data from A register to data location addressed by contents of DAR 3 |
| 108 | Instruction: Decrement all DARs and B register. If B register contains zero, set the EQ flip-flop |
| 109 | Instruction: If EQ = 1 stop. If EQ = 0, put 105 in the program counter |

nents will be needed to build the system.
3) Choose the best microprocessor for a system by looking at its bit length, speed, instruction set and timing features, to see if they will be suitable for the task at hand.
4) Connect the components together, using the timing and control signals from the processor to control the operation of the other components.
5) Microprocessors provide basic instructions to move data, and perform arithmetic, logical, comparison and branching operations. When these instructions are familiar and are combined correctly, the microprocessor can perform very complicated decisions and solve complex problems.
6) Develop a program with an organized, systematic and detailed approach. Break the program into many simple understandable tasks.
7) Write and verify the subprograms for each of these tasks and combine them into the required overall program.

### Microprocessor exercise

The previous two pages contain an exercise to illustrate the operation of microprocessors and the interaction between the microprocessor and memory. The data memory and the microprocessor's registers are shown in the centre, and the instructions are listed on the bottom of both pages.

You are to provide the microprocessors action: incrementing the program counter; fetching the next instruction; interpreting and implementing that instruction; and reading and writing into the data memory and the microprocessor's registers when needed.

As each instruction is executed, enter the results in the microprocessors registers and the relevant memory locations. The worksheet in the top left-hand side of the spread is for you to enter the contents of the various registers at each stage of the program. Assume that the A and B registers and memory locations can only hold a single decimal digit. The solution to this problem is shown on the right-hand page. Whenever an action changes the value of a number, it is shown in bold. As you work through the exercise, keep the solution covered with a piece of paper, then check your working later.

## Glossary

| | |
|---|---|
| **address** | an expression, usually numerical, which designates a specific location in a storage or memory device |
| **address format** | the arrangement of the address parts of an instruction |
| **direct addressing** | method of programming that has the address pointing to the location of data or the instruction that is to be used |
| **format** | the arrangement of data |
| **immediate address** | instruction in which an address part contains the value of an operand, rather than its address |
| **indexed address** | address that is added to the contents of an index register prior to or during the execution of a computer instruction. A variation on register indirect addressing |
| **instruction format** | the organisation and layout of an instruction |
| **register addressing** | operation involving an instruction that addresses a register as one of the data locations involved in the operation |
| **register indirect addressing** | like register addressing, but the address of data in memory is held in a register |